Conceptual Modeling for XML: A Survey

Martin Necasky

Charles University, Faculty of Mathematics and Physics,
Malostranske nam. 25, 118 00 Praha 1, Czech Republic
martin.necasky@mff.cuni.cz

Abstract

Recently XML is the standard format used for the exchange of data between information systems and is also frequently applied as a logical database model. If we use XML as a logical database model we need a conceptual model for the description of its semantics. However, XML as a logical database model has some special characteristics which makes existing conceptual models as E-R or UML unsuitable. In this paper, the current approaches to the conceptual modeling of XML data are described in an uniform style. A list of requirements for XML conceptual models is presented and described approaches are compared on the base of the requirements.

Keywords: conceptual modeling, XML, XML Schema

1 Introduction

Today XML is used for the exchange of data between information systems and it is frequently used as a logical database model for storing data into databases. If we use XML as a logical database model we need a conceptual model for modeling XML data. There is the Entity-Relationship (E-R) [25] model for the conceptual modeling of relational data. However, XML as a logical database model has some special differences which makes the E-R model unsuitable for the conceptual modeling of XML data. The main differences are the following:

- hierarchical structure
- irregular structure
- ordering on siblings
- mixed content

These features can not be properly modeled in the E-R model. There are some approaches, for example Extended E-R [1], EReX [16], EER [18], XER [23], ERX [22], and C-XML [9], trying to extend the E-R model to be suitable for the conceptual modeling of XML data. It is possible to extend the E-R model to model ordering, mixed content, and irregular structure of XML data. However, there is a problem with the modeling of a hierarchical structure of XML data.

Suppose an E-R diagram with a relationship type *Enroll* between two entity types *Student* and *Course* representing courses enrolled by students. Each student may enroll zero or more courses and each course may be enrolled by zero or more students. The diagram is shown in Figure 1.1(a).

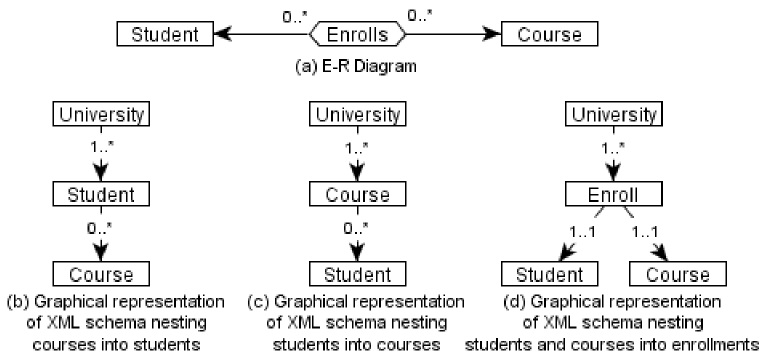
Student
0..*
Enrolls
0..*
Course
(a) E-R Diagram
University
1..*
Student
0..*
Course
(b) Graphical representation of XML schema nesting courses into students
University
1..*
Course
0..*
Student
(c) Graphical representation of XML schema nesting students into courses
University
1..*
Enroll
1..1
1..1
Student
Course
(d) Graphical representation of XML schema nesting students and courses into enrollments

Figure 1.1: Representation of E-R relationship type in a hierarchical structure

Figures 1.1(b), (c), and (d) show possible representations of the relationship type in a hierarchical structure. Oriented arrows denote a nesting. There is not the best nesting of the concepts. The nesting of courses into students illustrated by Figure 1.1(b) is suitable when we need to see students and the courses they enrolled. The nesting of students into courses illustrated by Figure 1.1(c) is suitable when we need to see courses and the students enrolled in them.

The previous example shows another difference between the conceptual level of XML and the E-R model. This difference is not in the structure but it is in the usage of XML. It is shown that there may be many ways of how to use entity types connected together by a relationship type. If we represent data in the form of XML, each of these ways may require another hierarchical ordering of the entities. However, this feature can not be effectively modeled by the E-R model.

Another possibility of how to model XML data is to start from a hierarchical structure. This approach may be called the hierarchical approach. There are conceptual models based on the hierarchical approach, for example X-Entity [15], ORA-SS [7], and Semantic Networks for XML [11]. The base of a schema in the hierarchical approach is a tree, whose nodes are entity types and edges are relationship types between the entity types. Figures 1.1(b), (c), and (d) show examples of a basic hierarchical schemata.

The hierarchical approach is able to solve the mentioned problem with different views of the same data. For each of the views there is a separate tree. However, a problem with the modeling of attributes of relationship types or with the modeling of n-ary relationship types, effectivelly solved in the E-R model, arises. Another problem arises when deciding which of hierarchical organizations of the same data is the best to select as the basic organization used for the data storage.

The goal of this paper is to describe the existing conceptual models for XML based on the E-R model and on the hierarchical approach. There are approaches based on the UML (Unified Modeling Language) [21] and ORM (Object Role

Modeling) [13] models, too. However, we do not describe them in this paper. We propose a list of requirements for conceptual models for XML and compare the described models against the requirements. The main contributions of this paper are the unified descriptions of the conceptual models and the comparison of the models against the list of requirements.

Section 2 introduces the list of requirements for conceptual models for XML. Section 3 describes languages for describing XML schemata on the logical level. Section 4 formally describes the well-known E-R model and conceptual models for XML based on the well-known E-R model. Section 5 describes hierarchical conceptual models for XML. Section 6 compares the described conceptual models against the requirements introduced in Section 2.

2 Requirements for Conceptual Models for XML

In this section, we summarize requirements for conceptual models for XML. There are two groups of the requirements described. The first group consists of *general requirements* covering general goals of the XML conceptual modeling. The second group consists of *modeling constructs requirements* covering requirements on what kinds of modeling constructs should XML conceptual models support.

2.1 General Requirements

Independence on XML schema languages The conceptual model should be independent on a certain XML schema language (XML Schema [12], DTD, ...). The constraints given by a certain XML schema language should not be propagated to the conceptual level. It should be a conceptual model for XML data, not a conceptual model for the structures of a certain XML schema language.

Formal foundations The modeling constructs of the conceptual model should be described formally, which allows to compare the model with other conceptual models or to describe the operations on the model structures and modeled data (for example, data transformation between two conceptual schemata or their integration).

Graphical notation A user-friendly graphical notation for the formal modeling constructs should be offered by the conceptual model.

Logical level mapping There should be algorithms for mapping of the conceptual modeling constructs to the XML logical level. The logical schema should implement as many integrity constraints arised from the conceptual schema as possible. It may require the usage of more than one XML schema language for the logical level description (XML Schema and Schematron [14], for example). The hierarchical structure of the XML data should be utilized as much as possible on the logical level.

Different structures on the logical level The XML logical level is hierarchical. However, there are different users with different requirements accessing the modeled data on the logical level. Hence, there can be different hierarchical views of the same data. Each of the views suits to different requirements. It should be possible to model the different hierarchical views on the conceptual level and translate them to the corresponding views on the logical level. Moreover, there should be algorithms allowing automatic translation of data from one logical view to another logical view (using XSLT [5], for example).

Semantic web mapping With the increasing usage of the semantic web technologies the problem of publishing data in the form of RDF [19] triples described by RDF Schema [19] or OWL [24] arises. One possible solution is to have the data internally represented in the form of XML and translate them to the RDF triples represented in the form of RDF/XML [19] utilizing XSLT. The conceptual model for XML should consider this problem. It would be useful to have algorithms for the translation from the conceptual level to the semantic web level where the structures from the conceptual level are described using OWL. It would allow companies to publish their internally represented data on the semantic web and, backwards, to obtain data from the semantic web and integrate them to the internal representation automatically.

2.2 Modeling Constructs Requirements

Hierarchical structure Although it can be useful to keep a document designer out of the hierarchical structure of XML data on the conceptual level, the conceptual model should offer modeling constructs for modeling nesting explicitly. For example, aggregation relationship types can be used. However, non-hierarchical relationship types (for example, association relationship types or references) should be offered too. The conceptual model should introduce contructs for modeling a recursive structure.

Cardinality for all participants The hierarchical structure of XML data restricts the specification of cardinality constraints only to the nested participants of the relationship type. However, it should be possible to specify cardinality constraints for the all participants on the conceptual level.

***N*-ary relationship types** For the same reason, the modeling of *n*-ary relationship types and their translation to the XML logical level is problematic. However, it should be possible to model *n*-ary relationship types on the conceptual level.

Attributes of relationship types For the same reason again, the modeling of attributes of relationship types is problematic. Nor the nesting nor the concept of referential integrity on the XML logical level do not allow to directly express attributes of relationship types. However, the conceptual model should allow to model attributes of relationship types.

Ordering XML is ordered and this property should be propagated to the conceptual level. It should be possible to express the ordering on values of attributes, the ordering on concepts connected with another concept (for example, a book has a title page first, followed by an abstract, chapters, appendixes and a bibliography in this order), and the ordering on a participant of a relationship type (for example, the list of authors of a book or the list of chapters of a book are ordered).

Irregular and heterogeneous structure XML data may have irregular and heterogeneous structure. The conceptual model should introduce constructs for modeling such a structure. For example, variant-valued constructors for constructing attributes or disjunctive relationship types should be introduced.

Document-centric data The difference between the conceptual models for XML and the other conceptual models is that the conceptual models for XML must allow to model document-centric data. It means that not only the real-world objects with attributes and relationships but also the certain parts of documents are modeled on the conceptual level. Hence, there should be corresponding modeling constructs offered by the conceptual model. It means to allow attributes and relationships of a given concept to be mixed with a text when represented in a document content. However, the mixed content should not be restricted as it is restricted by XML Schema. Some form of generalized mixed content should be introduced allowing to specify where the text values may appear exactly (as it is possible in Relax NG [6] schemata, for example).

Reuse of content The reuse of content should be supported by the conceptual model. For example, the concept inheritance (modeled by IS-A relationship types in E-R, for example) supports the reuse of content. However, the conceptual model may be inspired in the XML Schema language and may support named types and named groups of concepts on the conceptual level.

Integration of conceptual schemata XML data are often used for the data integration. However, it can not be done effectivelly and automatically without the support on the conceptual level. A conceptual model for XML should offer modeling constructs to support an integration of schemata on the conceptual level and it should allow to merge different conceptual schemata to an overall conceptual schema. Further, it would be useful to generate XSLT transformation scripts to translate data corresponding to one conceptual schema to data corresponding to another conceptual schema.

3 Schema Languages for XML

In this section, we describe schema languages for the description of XML data. Two groups of the schema languages are distinguished. The first group is called *tree grammar based XML schema languages* and the second group is called *tree pattern based XML schema languages.* The common XML schema languages as DTD or XML Schema are members of the first group. A member of the second group is Schematron, for example. The XSLT language can be comprehended as a member of the second group too.

3.1 Tree Grammar Based XML Schema Languages

In this section, we describe the common XML schema languages based on the tree grammars as restrictions of a more general tree grammar called *XGrammar.*

This notation was proposed by Mani et al. in [18]. XGrammar formalizes the most important features of existing XML schema languages as XML Schema, DTD, and RELAX. From RELAX, the authors borrow the notion of *tree* and *hedge* types: the values of a tree type are trees and the values of a hedge type are hegdes - sequences of trees.

The authors use G to denote a schema in XGrammar and $L(G)$ to denote the language that G generates. The existence of a set $\widehat{N}$ of non-terminal symbols, a set $\widehat{T}$ of terminal names and a set $\widehat{\tau}$ of atomic data types (such as string, integer, etc) including ID and $IDREF(S)$ is assumed.

Definition 3.1 (XGrammar) :
A *XGrammar* is denoted by a 7-tuple $G = (N_T, N_H, T, S, E, H, A)$ where:

- N_T is a set of non-terminal symbols that are tree types, where $N_T \subseteq \widehat{N}$,
- N_H is a set of non-terminal symbols that are hedge types, where $N_H \subseteq \widehat{N}$, $N = N_T \cup N_H$, $N_T \cap N_H = \emptyset$,
- T is a set of terminal symbols, where $T \subseteq \widehat{T}$,
- S is a set of start symbols, where $S \subseteq N$,
- E is a set of element production rules of the form $X \to a\,RE$, where $X \in N_T, a \in T$, and RE is:

$$\begin{aligned} RE \;::= & \;\epsilon \,|\, \tau \,|\, n \,|\, (RE) \,|\, (RE|RE) \,|\, (RE + RE) \,|\, (RE, RE) \,|\, (RE)^? \,| \\ & (RE)^* \,|\, (RE)^+, \end{aligned}$$

  where $\tau \in \widehat{\tau}$ and $n \in N$. Note that RE is actually a hedge type, but it might not have a name associated with it. In other words, we can have anonymous hedge types not captured by N_H.
- H is a set of hedge production rules of the form $X \to RE$, where $X \in N_H$, and RE is the same as the one for E,
- A is a set of attribute production rules of the form $X \to a\,RE$, where $X \in N, a \in T$, and RE is:

$$RE \;::=\; \epsilon \,|\, \alpha \,|\, (RE) \,|\, (RE, RE),$$

  where α is an attribute definition defined as:

$$\alpha \;::= \begin{cases} \text{"@"} \;\; a \;\; [\text{"?"}] \;\; \text{"::"} \;\; \tau & \text{if}\, \tau \notin \{IDREF, IDREFS\} \\ \text{"@"} \;\; a \;\; [\text{"?"}] \;\; \text{"::"} \;\; \tau \;\; \text{"}\rightsquigarrow\text{"} \; RE_1 & \text{if}\, \tau = IDREF \\ \text{"@"} \;\; a \;\; [\text{"?"}] \;\; \text{"::"} \;\; \tau \;\; \text{"}\rightsquigarrow\text{"} \; RE_2 & \text{if}\, \tau = IDREFS \end{cases}$$

  where $\tau \in \widehat{\tau}$ and

$$\begin{aligned} RE_1 \;::= & \; n_t \,|\, (RE_1) \,|\, RE_1 + RE_1, \quad \text{where} \quad n_t \in N_T \\ RE_2 \;::= & \; \epsilon \,|\, n \,|\, (RE_2) \,|\, (RE_2|RE_2) \,|\, (RE_2 + RE_2) \,|\, (RE_2, RE_2) \,|\, (RE_2)^? | \\ & (RE_2)^* \,|\, (RE_2)^+, \quad \text{where} \quad n \in N \end{aligned}$$

Assume the following example XML document. It describes an university department. It has one or more study fields and one or more professors. Each study field offers one or more courses. Each course consists of lessons and practices. Each professor leads zero or more courses and garants zero or one study fields.

```
<department name="dep1">
  <studyfield name="sf1">
    <course code="c1" name="Course 1">
      <lesson time="WS05-c1-1"/>
      <practice time="WS05-c1-2"/>
      <practice time="WS05-c1-3"/>
    </course>
    <course code="c2" name="Course 2">
      <lesson time="LS06-c2-1"/>
    </course>
    <course code="c3" name="Course 2">
      <practice time="LS06-c3-1"/>
      <practice time="LS06-c3-2"/>
      <practice time="LS06-c3-3"/>
    </course>
  </studyfield>
  <professor persnum="p1" office="o1">
    <leads courses="c1 c3"/>
  </professor>
  <professor persnum="p2" office="o2" />
  <professor persnum="p3" office="o2">
    <leads courses="c2"/>
    <garant studyfield="sf1"/>
  </professor>
</department>
```

The XML document is described by the following XGrammar $G = (N_T, \emptyset, T, S, E, \emptyset, A)$:

$$\begin{aligned}
N_T &= \{Department, StudyField, Course, Lesson, Practice, Professor, Leads,\\
&\quad Garant\}\\
T &= \{department, studyfield, course, lesson, practice, professor, leads,\\
&\quad garant, name, code, time, office, since, persnum\}\\
S &= \{Department\}
\end{aligned}$$

$$
\begin{aligned}
E =& \{Department \rightarrow department(StudyField^{+}, Professor^{+}),\\
& StudyField \rightarrow studyfield(Course^{+}),\\
& Course \rightarrow course(Lesson^{*}, Practice^{*}),\\
& Lesson \rightarrow lesson(\epsilon),\\
& Practice \rightarrow practice(\epsilon),\\
& Professor \rightarrow professor(Leads^{*}, Garant^{?}),\\
& Leads \rightarrow leads(\epsilon),\\
& Garant \rightarrow garant(\epsilon)\}\\
A =& \{Department \rightarrow department(@name :: ID),\\
& StudyField \rightarrow studyfield(@name :: ID),\\
& Course \rightarrow course(@code :: ID, name),\\
& Lesson \rightarrow lesson(@time),\\
& Practice \rightarrow practice(@time),\\
& Professor \rightarrow professor(@persnum :: ID, @office),\\
& Leads \rightarrow leads(@courses :: IDREFS \rightsquigarrow Course^{*}),\\
& Garant \rightarrow garant(@studyfield :: IDREF \rightsquigarrow StudyField, @since)\}
\end{aligned}
$$

In [17] Mani proposes a more general form of a tree grammar called *regular tree grammar* and two restrictions of this grammar called *local tree grammar* and *single type tree grammar.* These restrictions can be specialized for the XGrammar. First, a competition of non-terminals that are tree types is defined.

Definition 3.2 (Competition of non-terminals) :
Let G be a schema in XGrammar and A and B be two different non-terminals which are tree types. A and B are said to be *competing* with each other if:

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

Definition 3.3 (Local XGrammar) :
A *local XGrammar* schema is a XGrammar schema without competing non-terminals.

Definition 3.4 (Single-type XGrammar) :
A *single-type XGrammar* schema is a XGrammar schema such that:

- for each element production rule, non-terminals in its content model do not compete with each other, and
- start symbols do not compete with each other.

Mani in [17] describes common XML schema languages as regular tree grammars. DTD is a local XGrammar. This is enforced by not distinguishing between terminals and non-terminals. There is one and only one element production rule $E \rightarrow e\, RE$ for each non-terminal symbol $E \in N_T$ in a DTD schema and for any other element production rule $F \rightarrow f\, RE$ in the DTD schema $e \neq f$ is valid.

The expressiveness of XML Schema is mostly within single-type and that was the intention of the specification. However, in some cases it fails to be in single-type. Mani in [17] describes how the main features of XML Schema can be described as single-type grammar. In the case of RELAX NG, Mani states that any regular tree grammar can be expressed in RELAX NG.

3.2 Tree Pattern Based XML Schema Languages

Using a tree grammar based XML schema language a document engineer creates a whole grammar according to top-down production rules in a specified formalism. He or she describes the required structure of documents and can add some data types, key, and referential integrity constraints.

However, not all the required constraints which should be satisfied by a document can be expressed in a tree grammar based language. For example, there are functional dependencies and structural constraints which are hard or impossible to express in a tree grammar based XML schema language. These constraints can be a result of the conceptual modeling of XML data. However, they can not be expressed in this kind of languages.

The following functional dependencies and structural constraints are problematic when describing XML data only by tree grammar based languages.

a) All the elements described by the path $/projects/project/professor$ having the same value of their subelement $profid$ have the same value of their subelements *name* and *email*. This constraint is a functional dependency. There are subelements *paper* representing the papers written by the professor during his work in a project. These sublements are not constrained in this way, because a professor can work in more projects and in each of the projects he wrotes different papers.

b) Each of the elements described by the path */projects/project* must have the subelement *controlled* if it has the subelement *sponsored*. This constraint is a structural constraint. If a project has a sponsor, it must be controlled by someone. If a project does not have any sponsor, it does not have to be controlled.

As the answer to this problems, another family of XML schema languages called the *tree pattern based XML schema languages* was developed. These languages are based on the idea of specifying rules for documents. They are built as an XML envelope of the XSLT language which can be conceived as a tree pattern based language too. The navigation through documents is realized by the XPath language.

There are two representatives of the tree pattern based XML schema languages. The first is called *SchemaPath* introduced by Marinelli et al. in [20] and the second is called *Schematron* specified in [14]. The both languages are XML based languages.

SchemaPath extends the XML Schema language with just one new construct and one new build-in type. Schematron is a new language which has nothing to do with XML Schema. It can be easily transformed into an equivalent XSLT document. Hence, an XSLT processor can be used as a validator for Schematron schemata. The languages are not further described in this paper. As the demonstration of the power of tree pattern based XML schema languages we introduce the descriptions of the functional dependency from a) and the structural constraint from b). The structural constraint can be easilly described using Schematron as follows:

```
<rule context="/projects/project">
  <assert test="not(sponsored) or
                  (sponsored and controlled)">
  Structural constraint violated.
  </assert>
</rule>
```

The functional dependency can not be described by one XPath predicate in the test attribute. However, it can be described using XSLT.

```
<xsl:template match="/projects/project/professor">
  <xsl:variable name="prof">
    <xsl:value-of select=".">
  </xsl:variable>
  <xsl:for-each select=
        "/projects/project/professor[profid=$prof/profid]">
    <xsl:if test="not(name=$professor/name and
        email=$professor/email)">
      Functional dependency violated.
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

4 E-R Based Conceptual Models for XML

In this section, we describe several conceptual models for XML based on the E-R model. First, we introduce the well-known E-R model. The rest of the section describes current models for modeling XML data based on the E-R model. These models are: Extended E-R [1], EReX [16], EER [18], XER [23], ERX [22], and C-XML [9].

4.1 E-R Model

In this section, we formally describe the well-known E-R model. The formalism used here was proposed by Thalheim in [25]. First, *data schemata* and *tuple functions* are defined.

Each E-R schema has its data schema. The data schema contains a set of simple attributes used for composing entity and relationship types, a set of domains containing possible values of attributes and a domain function assigning a domain to each attribute in the data schema. Data schemata are formally defined by the following definition.

Definition 4.1 (Data schema) :
A *data schema* $DD = (U, D, dom)$ is given by a finite set U of *simple attributes* $\{A_1, A_2, \ldots\}$, by a set $D = \{D_1, D_2, \ldots\}$ of *domains*, and by a *domain function* $dom : U \rightarrow D$ which associates every attribute with its domain.

Tuple functions are used as a formalization of instances of the entity types modeled in an E-R schema. They are defined as follows.

Definition 4.2 (Tuple function) :
Let $DD = (U, D, dom)$ be a data schema. Let $D_{DD} = \bigcup_{A \in U} dom(A)$. A *tuple* on $X \subseteq U$ and on DD is a function

$$t : X \to D_{DD}$$

with $t(A) \in dom(A)$ for $A \in X$, where $t(A)$ is called the *value of the attribute* A. If the set X is linear ordered, i.e. $X = \{A_1, \ldots, A_m\}$ with $A_1 \leq \cdots \leq A_m$, then the tuple t is denoted by

$$(t(A_1), \ldots, t(A_m))$$

Entity types *Entity types* represent real world objects modeled in an E-R schema. Each entity type has its *extension* containing *entities*. Each entity represents one real world object. There are *strong entity types* and *weak entity types*. The strong entity types are defined by the following definition.

Definition 4.3 (Strong entity type) :
A *strong entity type* has the form

$$E = (attr(E), id(E))$$

where E is the name of the entity type, $attr(E)$ is a set of simple attributes from U and $id(E)$ is a non-empty subset of $attr(E)$ called the *key* of the entity type.

A strong entity type has a key. An entity of a strong entity type is identifiable by a value of the key of the entity type. On the other hand, a weak entity type depends on other strong/weak entity types. The key of a weak entity type is composed of its own key and the keys of the entity types it depends on. We define weak entity types depending on one entity type in the following definition. It can be extended to the general case, where a weak entity type depends on more than one entity type.

Definition 4.4 (Weak entity type) :
A *weak entity type* has the form

$$E = (E', attr(E), id(E))_{weak},$$

where E is the name of the entity type, $attr(E)$ is a set of simple attributes from U, $id(E)$ is a non-empty subset of $attr(E)$ called the *partial key* of the entity type and E' is the *identification entity type* which the entity type depends on. The key of the weak entity type is $id(E) \cup id(E')$.

A strong entity type is displayed by a box with the entity type name in the middle of the box. A weak entity type is displayed by a box with an inner diamond connected by a solid arrow to its identification entity type. Attributes are displayed by circles connected by a solid line with their entity types. A name of an attribute is displayed at the attribute's circle. A key attribute is displayed by a filled circle.

Figure 4.1 illustrates strong and weak entity types. The problem is to model persons and their addresses changing in time. However, the history of person's addresses must be stored. The problem can be solved by creating a strong entity type named $Person$ and a weak entity type named $PAddress$. The entity type $Person$ has attributes $personnum$ and $name$ where $personnum$ is the key of $Person$, i.e. $attr(Person) = \{personnum, name\}$ and $id(Person) = \{personnum\}$. The entity type $PAddress$ has attributes $from$, to, $street$, and $city$ where $from, to$ is the partial key of $PAddress$, i.e. $attr(PAddress) = \{from, to, street, city\}$ and $id(PAddress) = \{from, to\}$. The schema is formally described as follows.

$$\begin{aligned} Person &= (\{persnum, name, \}, \{persnum\}), \\ PAddress &= (Person, \{from, to, street, city\}, \{from, to\})_{weak} \end{aligned}$$

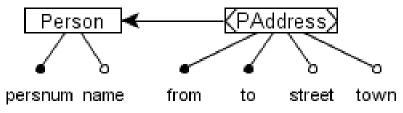

Figure 4.1: E-R Diagram - Strong and Weak Entity Types

Next, the extensions of entity types are defined. An extension of an entity type is a set containing concrete *entities* of the entity type. Entities are defined as tuples on attributes of the entity type. An extension of a strong entity type is defined as follows.

Definition 4.5 (Extension of a strong entity type) :
An *extension* of a strong entity type $E = (attr(E), id(E))$ is a set E^C of tuples $\{e_1, \ldots, e_n\}$ on $attr(E)$ satisfying the key condition:

$$(\forall\, e, e' \in E^C)\,(\exists\, A \in id(E))\,(e(A) \neq e'(A))$$

The definition of an extension of a weak entity type is similar to the definition of an extension of a strong entity type. However, the existence condition must be satisfied, i.e. there must be a corresponding entity of an entity type the weak entity type depends on.

Definition 4.6 (Extension of a weak entity type) :
An *extension* of a weak entity type $E = (E', attr(E), id(E))_{weak}$ is a set E^C of tuples $\{e_1, \ldots, e_n\}$ on $attr(E) \cup id(E')$ satisfying the key condition:

$$(\forall\, e, e' \in E^C)\,(\exists\, A \in id(E) \cup id(E'))\,(e(A) \neq e'(A))$$

Moreover, the existence condition must be satisfied:

$$(\forall\, e \in E^C)\,(\exists\, e' \in E'^C)\,(\forall\, A \in id(E'))\,(e(A) = e'(A))$$

For the schema displayed in Figure 4.1 the following extensions are valid.

$$\begin{aligned}
Person^C = & \{(\text{'p1', 'John Black'}),\\
& (\text{'p2', 'Jane White'})\}\\
PAddress^C = & (\text{'2000-07-25', '2004-01-10', 'Broadway Av', 'West Beach', 'p1'}),\\
& (\text{'2004-01-10', '2005-12-02', 'Pazmaniteng', 'Vienna', 'p1'}),\\
& (\text{'1995-04-13', '2005-12-02', 'Hoge Wei', 'Zaventem ', 'p2'})\}
\end{aligned}$$

Relationship types The next important modeling construct of the E-R model is a relationship type construct. Relationship types represent associations between real world objects. A relationship type is constituted by a list of entity types connected by the relationship type and a list of attributes of the relationship type. A role can be specified for each entity type in the relationship type.

Definition 4.7 (Relationship type) :
A *relationship type* has the form

$$R = (compon(R), card(R), attr(R)),$$

where R is the name of the relationship type and $attr(R)$ is a set of simple attributes from U. The member $compon(R) = (l_1 : E_1, \ldots, l_n : E_n)$ is a sequence of entity types prefixed by labels from a set of strings L. A label may be empty. Each entity type E_i is called a *participant* in R and each non-empty l_i is called the *role* of the participant E_i in the relationship type R, $1 \leq i \leq n$. Roles of participants in R are pairwise distinct. The member $card(R) = (c_1, \ldots, c_n)$ is a sequence of *cardinality constraints*. The member $c_i = (min_i, max_i)$ is a cardinality constraint for the participant E_i, $1 \leq i \leq n$.

A relationship type is displayed by a diamond with solid arrows leading to its participants. Each of the arrows is labeled by the role and the cardinality constraint of the participant connected by the arrow. Each attribute of the relationship type is connected with the diamond by a solid line. The name of the relationship type is displayed in the middle of the diamond.

Figure 4.2 displays an E-R schema with strong entity types $Course$ and $Student$, and relationship types $Enrolls$ and $IsPrerequisite$ between them. The relationship type $Enrolls$ is a binary relationship type between entity types $Student$ and $Course$, and represents students enrolled in courses. It has the attribute $result$ representing the result of a given student enrolled in a given curse. There is the cardinality constraint $(0, *)$ for $Student$ and the cardinality constraint $(1, *)$ for $Course$. The relationship type $IsPrerequisite$ is a recursive relationship type. In the case of a recursive relationship type, roles of the participants should be used to distinguish them. For example, there is the entity type $Course$ in the role $requires$ with the cardinality constraint $(0, *)$ and in the role $required$ with the cardinality constraint $(0, *)$. The schema is formally described as follows.

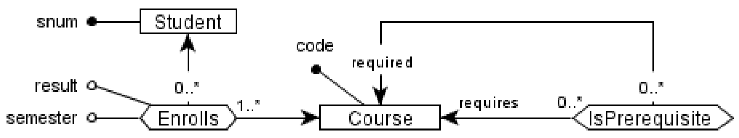

Figure 4.2: E-R Diagram - Relationship Types

$$\begin{aligned}
Student &= (\{snum\}, \{snum\}),\\
Course &= (\{code\}, \{code\}),\\
Enrolls &= ((Student, Course), ((0,*), (1,*)), \{result, semester\}),\\
IsPrerequisite &= ((required : Course, requires : Course), ((0,*), (0,*)), \emptyset)
\end{aligned}$$

An extension of a relationship type is a subset of the cartesian product defined on the extensions of the participants and domains of the attributes of the relationship type.

Definition 4.8 (Extension of a relationship type) :
An *extension* of a relationship type $R = ((E_1, \ldots, E_n), (c_1, \ldots, c_n), \{A_1, \ldots, A_k\})$ is the set

$$R^C \subseteq E_1^C \times \ldots \times E_n^C \times dom(A_1) \times \ldots \times dom(A_k)$$

The following condition must be satisfied for each $1 \leq i \leq n$ where $c_i = (min_i, max_i)$:

$$(\forall\, e \in E_i^C)\,(min_i \leq |\{r \in R^C : r(E_i) = e\}| \leq max_i)$$

The members of R^C are called *relationships* of the relationship type R.

Let $Student^C = \{s_1, s_2, s_3\}$ and $Course^C = \{c_1, c_2, c_3\}$ be extensions of the entity types $Student$ and $Course$ from the schema displayed in Figure 4.2. The following extensions are valid:

$$\begin{aligned}
Enrolls^C = \;&\{(s_1, c_1, \text{'A'}, \text{'AS05'}),\\
&(s_1, c_2, \text{'B'}, \text{'AS05'}),\\
&(s_2, c_2, \text{'A'}, \text{'SS06'}),\\
&(s_2, c_3, \text{'C'}, \text{'SS06'})\}\\
IsPrerequisite^C = \;&\{(c_2, c_1),\\
&(c_3, c_2)\}
\end{aligned}$$

An extension of a role of an entity type E participating in a relationship type can be defined as a subset of the extension of the entity type E.

Definition 4.9 (Extension of a role) :
Let R be a relationship type with a participant $l : E$. An *extension* of the role l of the entity type E in the relationship type R is the set $R.l^C$ of all $e \in E^C$ such that

$$(\exists\, r \in R)\,(e = r(l : E))$$

Assuming $IsPrerequisite^C$ from the previous example, the extensions of the roles *requires* and *required* of the entity type *Course* in the relationship type *IsPrerequisite* are the following.

$$IsPrerequisite.requires^C = \{c_1, c_2\}$$
$$IsPrerequisite.required^C = \{c_2, c_3\}$$

IS-A Relationship Types IS-A relationship types are a special kind of binary relationship types. They are used for modeling specialization and generalization of entity types. IS-A relationship types are defined by the following definition.

Definition 4.10 (IS-A relationship type) :
An *IS-A relationship type* has the form

$$(E_1, E_2)_{IS-A}$$

where $E_1 = (attr(E_1), id(E_1))$ and $E_2 = (attr(E_2), id(E_2))$ are entity types. The IS-A relationship type defines a subtype hierarchy, i.e.

$$attr(E_1) \subseteq attr(E_2)$$

and a subset hierarchy, i.e.

$$E_2^C \subseteq E_1^C$$

The entity type E_1 is called the *general entity type* and E_2 is called the *special entity type* of the IS-A relationship type.

A specialization of an entity type E_1 is an entity type E_2 having the attributes and the key of E_1, and some additional attributes. Moreover, it may have an additional key. The subset hierarchy in the definition implies that each entity e in E_2 is an entity in E_1 (the tuple e is restricted to the attributes of E_1).

An IS-A relationship type is displayed by an non-filled arrow going from its special entity type to its general entity type. Let $(E_1, E_2)_{IS-A}$ be an IS-A relationship type. The attributes from $attr(E_1)$ of the entity type E_2 are not displayed at the box of E_2. They are assumed to be attributes of E_2 implicitly. Only the attributes from $attr(E_2) \setminus attr(E_1)$ are displayed at the box of E_2.

Figure 4.3 displays the specialization of the entity type *Person* to the entity types *Student* and *Professor* designed via IS-A relationship types. The schema is formally described as follows.

$$\begin{aligned} Person &= (\{persnum\}, \{persnum\}), \\ Student &= (\{snum\}, \{snum\}), \\ Professor &= (\{office(building, room)\}, \emptyset) \end{aligned}$$

$$(Person, Student)_{IS-A},$$
$$(Person, Professor)_{IS-A}$$

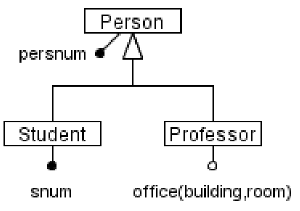

Figure 4.3: E-R Diagram - IS-A Relationship Types

Complex Attributes The concept of optional, composite or multivalued attributes can be considered. This concept can be naturally transformed to XML schema languages. The integral view of these kinds of attributes is given by the following definition proposed by Thalheim in [25]. The author proposes the constructors for creating tuple and set-valued complex attributes. The following definition extends the notion of complex attributes with constructors for creating bag, list, and variant-valued complex attributes.

Definition 4.11 (Complex attributes) :
Let $DD = (U, D, dom)$ be a data schema and CA be a set of names different from U. The set UC of complex attributes with the empty word λ is defined as follows:

- $\lambda \in UC$
- $U \subseteq UC$
- If $X_1, \ldots, X_n \in UC$ are distinct complex attributes and $X \in CA$ then
$$X(X_1, \ldots, X_n) \in UC$$
is a tuple-valued complex attribute named X.
- If $X' \in UC$, $X \in CA$ and $m, n \in \{*\} \cup \{0, 1, \ldots\}$ then
$$X\{X'\}[m, n] \in UC$$
is a set-valued complex attribute named X.
- If $X' \in UC$, $X \in CA$ and $m, n \in \{*\} \cup \{0, 1, \ldots\}$ then
$$X\{|X'|\}[m, n] \in UC$$
is a bag-valued complex attribute named X.
- If $X' \in UC$, $X \in CA$ and $m, n \in \{*\} \cup \{0, 1, \ldots\}$ then
$$X\langle X'\rangle[m, n] \in UC$$
is a list-valued complex attribute named X.
- If $X_1, \ldots, X_n \in UC$ are distinct complex attributes then
$$X(X_1| \ldots |X_n) \in UC$$
is a variant-valued complex attribute named X.

A complex attribute is displayed by a circle as in the case of simple attributes. However, not only the name but the whole formal description of the attribute is displayed at the circle.

The set, bag, list or variant-valued complex attribute can be constructed as anonymous, i.e. a name X can be omited. For example, instead of the definition

$$name(titles\langle title\rangle[0,*], first, family)$$

the definition

$$name(\langle title\rangle[0,*], first, family)$$

can be used.

Most of the conceptual models for XML described in this paper propose their own complex attributes which are a subset of complex attributes from Definition 4.11. To ensure the consistency, we define complex attributes of each of the described models in the same way as the extension of simple attributes from Definition 4.1 to complex attributes from Definition 4.11.

The semantics of the constructors introduced in Definition 4.11 is given by the following definition. The definition extends the function *dom* to the function *Dom* defined on UC.

Definition 4.12 (Dom function) :
The $dom : U \to D$ function is extended to the $Dom : UC \to D$ function as follows:

- $Dom(\lambda) = \emptyset$.
- $\forall A \in U : Dom(A) = dom(A)$.
- For $X(X_1, \ldots, X_n) \in UC$, $Dom(X) = Dom(X_1) \times \ldots \times Dom(X_n)$.
- For $X\{X'\}[m,n] \in UC$, $Dom(X) = \mathcal{P}_m^n(Dom(X'))$ where $\mathcal{P}_m^n(M) = \{M' \subseteq M : m \leq |M'| \leq n\}$.
- For $X\{|X'|\}[m,n] \in UC$, $Dom(X) = \{(x_1, \ldots, x_k) : m \leq k \leq n \wedge (\forall 1 \leq i \leq k)(x_i \in Dom(X'))\}$ where $(x_1, \ldots, x_k)$ denotes an unordered list of items (not necessarily distinct).
- For $X\langle X'\rangle[m,n] \in UC$, $Dom(X) = \{\langle x_1, \ldots, x_k\rangle : m \leq k \leq n \wedge (\forall 1 \leq i \leq k)(x_i \in Dom(X'))\}$ where $\langle x_1, \ldots, x_k\rangle$ denotes an ordered list of items (not necessarily distinct).
- For $X(X_1|\ldots|X_n) \in UC$, $Dom(X) = Dom(X_1) \cup \ldots \cup Dom(X_n)$.

Figure 4.4 displays an example E-R schema. The schema represents professors, students, and courses at the university. This real-world objects are modeled by the following entity and relationship types.

$$
\begin{aligned}
Person &= (\{persnum, name(<title>, first, second),\\
&\qquad address(zip, town, street(name, no))\}, \{persnum\}),\\
Student &= (\{snum\}, \{snum\}), (Person, Student)_{IS-A},\\
Professor &= (\{office(building, room)\}, \emptyset), (Person, Professor)_{IS-A},\\
Thesis &= (\{title, type, year\}, \{title\}),\\
Department &= (\{name, \{phone\}[1,*]\}, \{name\}),\\
StudyField &= (\{name\}, \{name\}),\\
Course &= (\{code, name\}, \{code\}),\\
Lesson &= (Course, \{time(semester, day, start, end)\}, \{time\}),\\
Practice &= (Course, \{time(semester, day, start, end)\}, \{time\}),
\end{aligned}
$$

$$
\begin{aligned}
LeadsT &= ((Professor, Student, Thesis), ((0,*), (0,*), (1,1)), \emptyset),\\
Garant &= ((Professor, StudyField), ((0,1), (1,1)), \{since\}),\\
In &= ((Professor, Department), ((1,1), (1,*)), \emptyset),\\
Studies &= ((Student, StudyField), ((1,1), (1,*)), \emptyset),\\
Ensures &= ((Department, StudyField), ((1,*), (1,1)), \emptyset),\\
Enrolls &= ((Student, Course), ((0,*), (1,*)), \{result, semester\}),\\
LeadsC &= ((Professor, Course), ((0,*), (1,1)), \emptyset),\\
TeachesL &= ((Professor, Lesson), ((0,*), (1,1)), \emptyset),\\
TeachesP &= ((Professor, Practice), ((0,*), (1,1)), \emptyset),\\
Offers &= ((StudyField, Course), ((1,*), (1,*)), \emptyset),\\
IsPrerequisite &= ((required : Course, requires : Course), ((0,*), (0,*)), \emptyset)
\end{aligned}
$$

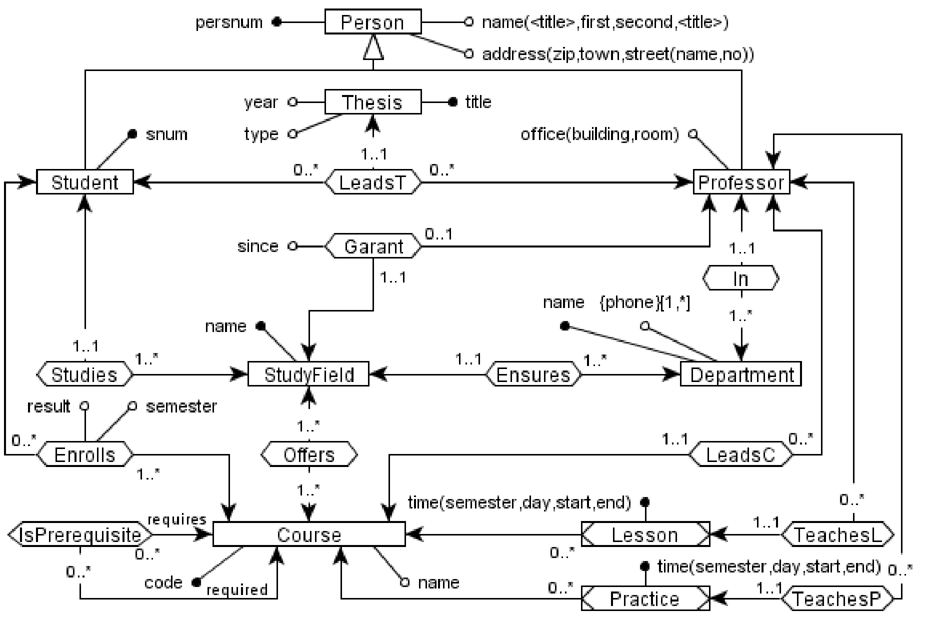

Figure 4.4: E-R Diagram - University

The modeling constructs of the well-known E-R model and some of its extending constructs were described in this subsection. In the following subsections, we describe existing extensions of the E-R model for the conceptual modeling of XML. We describe them following the formalism used for the description of the E-R model. Hence, we can be compare them with the well-known E-R model or compare them with each other.

4.2 Extended E-R Model (by Antonio Badia)

Extended E-R model proposed by Badia in [1] is a minimalistic extension to the E-R model. The extension is based on the idea of integration of structured and semistructured data where an overall conceptual schema is needed. Moreover, the author proposes algorithms for the translation of E-R schemata to relational schemata and to DTD schemata. Further, he studies the utilization of combination of relational schemata and DTD schemata for a data representation. The author identifies a minimal set of extensions by a reverse-engineering process from the DTD model to the extended E-R model and studies how the semantics of the DTD model should be expressed in the extended E-R model.

The author proposes the following DTD based extensions to the E-R model.

- *optional* and *required* attributes,
- *choice* attributes

The author does not introduce any special kind of relationship types for modeling hierarchical structure explicitly. Modeling of ordering and document-centric data is not possible. On the other hand, it is possible to model irregular and heterogeneous data by the choice attributes and the category concept mentioned in the formal description of the model.

4.2.1 Formal Description

As it was stated, the author identifies a minimal set of extensions by a reverse-engineering process from the DTD model. However, only a restricted form of the DTD model is considered. Following the Definition 3.1 of XGrammar used for a description of DTD schemata, the author considers only the element production rules of the form $X \rightarrow a\, RE$, where $X \in N_T, a \in T$, and RE is:

$$\begin{aligned} RE &::= \epsilon \,|\, \tau \,|\, n \,|\, (RECH) \,|\, (RE, RE) \,|\, n^? \,|\, n^* \,|\, n^+ \\ RECH &::= n \,|\, (RECH|RECH) \end{aligned}$$

where $\tau \in \widehat{\tau}$ and $n \in N$.

There are four possibilities of how the element B can be contained in the element A. The author introduces the entity types A^E and B^E representing the elements A and B, respectively on the conceptual level and introduces the following relationship type between A^E and B^E with the cardinality constraint $(1, 1)$ for A^E to represent the nesting of B in A:

- If there is no mark at the element B in the production rule for A a relationship type between A^E and B^E with the cardinality $(1, 1)$ for B^E is used.
- If there is '?' mark at the element B in the production rule for A a relationship type between A^E and B^E with the cardinality $(0, 1)$ for B^E is used.
- If there is '$*$' mark at the element B in the production rule for A a relationship type between A^E and B^E with the cardinality $(0, *)$ for B^E is used.
- If there is '+' mark at the element B in the production rule for A a relationship type between A^E and B^E with the cardinality $(1, *)$ for B^E is used.
- If the element B is contained in the element A in the form $B|C$, the author uses the transcription to $(B?, C?)$. He states, that the category concept can be used. The category is comprehended as an union of underlying entity types, but it is not specified more formally.

Hence, all possible relationships between elements which may be defined in the DTD model can be modeled by the modeling constructs of the E-R model and by the category concept. Futher, the author proposes two additional concepts.

The first concept is the concept of optional and required attributes. Every attribute in an extended E-R model must be marked as *required* or *optional.* If an attribute of an entity type or a relationship type is marked as optional, an entity or a relationship may not have a value of the attribute. It is different from the situation when the value of the attribute is empty.

The second concept is the concept of *choice* attributes. A choice attribute consists of two or more underlying attributes. For an entity type E a choice of attributes a and b means that an entity e from the extension of E has a value for the attribute a or a value for the attribute b. Further, the choice may be marked as *exclusive* or *inclusive*. The exlusive mark means that the entity e may have only a value of the attribute a or b but not the both. The inclusive mark means that the entity e may have values of the both attributes together.

Definition 4.13 (Choice and optional attributes) :
Given a data schema $DD = (U, D, dom)$ the set U_{oc} of simple, optional and choice attributes is defined as follows:

- $U \subseteq U_{oc}$
- If $X \in U$ then $X^? \in U_{oc}$ is an *optional attribute.*
- If $X_1, \ldots, X_n \in U$ then $(X_1|\ldots|X_n)_e \in U_{oc}$ is an *exlusive choice attribute.*
- If $X_1, \ldots, X_n \in U$ then $(X_1|\ldots|X_n)_i \in U_{oc}$ is an *inclusive choice attribute.*

In a graphical representation, an optional attribute is connected to the corresponding entity type by a solid line with two dashes crossing it. Required attributes are connected as they are now. A choice of attributes is expressed by marking the choice with an upward triangle, with the choices in the opposed side of the triangle. The author does not distinguish between exclusive and inclusive choice attributes in the graphical representation. However, it can be distinguished by displaying the symbol 'I' for inclusive and 'E' for exclusive choice attributes in the middle of the triangle, for example.

The semantics of the introduced kinds of attributes is given by the following definition. The definition extends the function *dom* to the function Dom_{oc} defined on U_{oc}.

Definition 4.14 (Domain function extension to U_{oc}) :
The $dom : U \to D$ function is extended to the $Dom_{oc} : U_{oc} \to D$ as follows:

- $\forall\, A \in U : \; Dom_{oc}(A) = dom(A)$
- For $X^? \in U_{oc}$, $Dom_{oc}(X^?) = \{\emptyset\} \cup \bigcup_{x \in Dom_{oc}(X)} \{\{x\}\}$.
- For $(X_1|\ldots|X_n)_e \in U_{oc}$, $Dom_{oc}((X_1|\ldots|X_n)_e) = \bigcup_{1 \leq i \leq n}(Dom_{oc}(X_i))$
- For $(X_1|\ldots|X_n)_i \in U_{oc}$, $Dom_{oc}((X_1|\ldots|X_n)_i) = Dom_{oc}(X_1^?) \times \ldots \times Dom_{oc}(X_n^?)$

An optional attribute $X^?$ is a set-valued complex attribute $\{X\}[0,1]$. An exclusive choice attribute $(X_1|\ldots|X_n)_e$ is a variant-valued complex attribute $(X_1|\ldots|X_n)$. An inclucive choice attribute $(X_1| \ldots |X_n)_i$ is a tuple-valued complex attribute $(\{X_1\}[0,1], \ldots, \{X_m\}[0,1])$.

Figure 4.5 displays the entity type *Student* having the optional attribute $phone^?$ and the choice attribute $(hostel(name, room)|home(street, city))_e$, i.e. each student has a hostel address or a home address but not both. On the left side, there are the attributes displayed in the author's style. On the right side, there are the attributes displayed in the formal style.

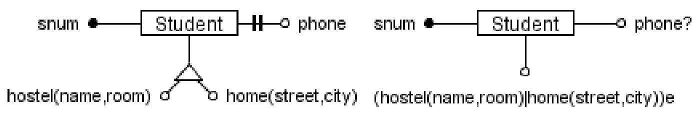

Figure 4.5: Extended E-R Diagram - Choice and Optional Attributes

4.2.2 Translation to XML Schema Languages

The author proposes algorithms for the translation from the extended E-R model to the relational model and the DTD model. He describes the algorithms in detail in [1].

The algorithm for the translation to the relational model uses the common techniques for the represention of optional attributes and choices in the relational model.

The main idea of the algorithm for the translation to the DTD model is the following. One of the entity types from a schema is selected and translated to the root element. All the other entity types are translated to child elements of the root element. This approach leads to flat XML documents with many references across the elements in them.

The author solves the problem whether to represent an entity type in the relational way or in the semistructured way. He utilizes the fact that today databases are able to store relational and semistructured data together. Hence, the combination of the two approaches can be utilized. Each entity type is separated in two parts: the relational part will take all the required attributes, and the semistructured part will take all the optional and choice attributes.

4.3 EReX

EReX is an extension to the E-R model proposed by Mani in [16], [17]. The author introduces the following extensions to the E-R model:

- structural specification called *categories*,
- constraints specifications called *coverage constraints* and *order constraints*

The concept of a special kind of relationship types for modeling hierarchical structure explicitly is not introduced by the author. Modeling of document-centric data is not possible too. On the other hand, modeling of ordering and irregular and heterogeneous data is possible with utilizing the extensions. However, only the ordering on the participant of the relationship type is possible.

4.3.1 Formal Description

The extensions are formally described by the author in his paper. However, we modified the formalism for the purposes of this paper.

First, the category relationship types are defined. The category relationship types are a special kind of binary relationship types similar to the IS-A relationship types.

Definition 4.15 (Category relationship type) :
A *category relationship type* has the form

$$(E_1, E_2)_{cat}$$

where $E_1 = (attr(E_1), id(E_1))$ and $E_2 = (attr(E_2), id(E_2))$ are entity types. The category relationship type defines a subtype hierarchy, i.e.

$$attr(E_1) \subseteq attr(E_2)$$

and a subset hierarchy, i.e.

$$E_2^C \subseteq E_1^C$$

The entity type E_1 is called the *categorized entity type* and E_2 is called the *category entity type* or *category* of the category relationship type. The categorized entity type may have an empty key. The categorizied entity type with an empty key must be categorized. The categorization must be constrained by total and exclusive coverage constraints.

A category relationship type is displayed by an arrow with the label CAT going from its category entity type to its categorizied entity type.

Total and *exclusive coverage constraints* can be specified for categories and for roles. A total coverage constraint specifies that the union of extensions of all included categories or roles must be the same as an extension of the categorizied entity type or the entity type with the included roles. The total coverage constraints are defined by the following two definitions.

Definition 4.16 (Total coverage constraint for categories) :
Let E be an entity type and $E_1, \ldots, E_n$ be category entity types of E. The *total coverage constraint* for $E_1, \ldots, E_n$, denoted as

$$E_1 + \cdots + E_n = E$$

specifies the following condition on $E^C, E_1^C, \ldots, E_n^C$:

$$E_1^C \cup \ldots \cup E_n^C = E^C$$

Definition 4.17 (Total coverage constraint for roles) :
Let E be an entity type and $R_1, \ldots, R_n$ be relationship types with a participant E. Let E participates in R_k in a role l_k, $1 \leq k \leq n$. The *total coverage constraint* for the roles $l_1, \ldots, l_n$ of E in $R_1, \ldots, R_n$, denoted as

$$R_1.l_1 + \cdots + R_n.l_n = E$$

specifies the following condition on $E^C, R_1.l_1^C, \ldots, R_n.l_n^C$:

$$R_1.l_1^C \cup \ldots \cup R_n.l_n^C = E^C$$

An exclusive coverage constraint specifies the disjunction between the extensions of the included categories or roles.

Definition 4.18 (Exclusive coverage constraint for categories) :
Let E be an entity type and $E_1, \ldots, E_n$ be category entity types of E. The *exclusive coverage constraint* for $E_1, \ldots, E_n$, denoted as

$$E_1 | \cdots | E_n$$

specifies the following condition on $E_1^C, \ldots, E_n^C$:

$$(\forall\, i, j,\ 1 \leq i, j \leq n,\ i \neq j)\,(E_i^C \cap E_j^C = \emptyset)$$

Definition 4.19 (Exclusive coverage constraint for roles) :
Let E be an entity type and $R_1, \ldots R_n$ be relationship types with a participant E. Let E participates in R_k in a role l_k, $1 \leq k \leq n$. The *exclusive coverage constraint* for the roles $l_1, \ldots l_n$ of E in $R_1, \ldots, R_n$, denoted as

$$R_1.l_1 | \cdots | R_n.l_n$$

specifies the following condition on $R_1.l_1^C, \ldots, R_n.l_n^C$:

$$(\forall\, i, j,\ 1 \leq i, j \leq n,\ i \neq j)\,(R_i.l_i^C \cap R_j.l_j^C = \emptyset)$$

The coverage constraints are not displayed in a graphical representation of a schema. They must be defined as additional integrity constraints.

The order constraints are specified for participants of a relationship type. They are defined as follows.

Definition 4.20 (Order constraint) :
Let R be a relationship type with participants $E_1, \ldots, E_n$. An *ordering* on a participant $E_i, 1 \leq i \leq n$, denoted by parenthesis $<$ and $>$ around E_i in the definition of R, specifies that for the given tuple $e \in E_i^C$ the set of relationships

$$R'^C = \{r \in R^C : (r(E_i) = e)\}$$

is linear ordered.

An ordering on a participant E of a relationship type R is displayed by a thick solid line between R and E.

Figure 4.6 displays the categorized entity type $Person$ and its categories $Student$ and $Professor$. The key of $Person$ is empty. Further, there are the entity types $Book$ and $Paper$ connected with $Professor$ by the relationship types $AuthorOfB$ and $AuthorOfP$, respectively. Attributes of $Book$ and $Paper$ are not displayed. There is an ordering specified on the entity types $Book$ and $Paper$ in the relationship types $AuthorOfB$ and $AuthorOfP$, respectively. It means, that the authors of a given paper or a given book are ordered.

The total coverage constraint $Student + Professor = Person$ specifies, that each person is a student or a professor and there are no other persons. The exclusive coverage constraint $Student|Professor$ specifies that students and professors are disjoint. The total coverage constraint $AuthorOfB.pbook + AuthorOfP.ppaper = Professor$ specifies that each professor is an author of some paper or book. If $pbook : AuthorOfB|ppaper : AuthorOfP$ constraint would be specified it would mean that a professor writes only books or only papers but not both.

4.3.2 Translation to XML Schema Languages

The author introduces an algorithm for the translation of EReX schemata to XGrammar language. The algorithm is formally described in [16]. In [17] more possibilities of translation are described. An advantage of the algorithm is that it maximizes the utilization of nesting in XML data and makes use of union and recursive types.

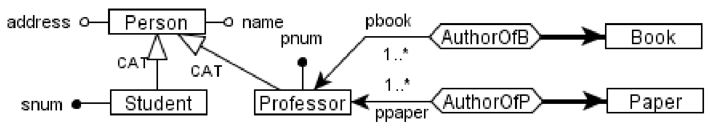

Figure 4.6: EReX Diagram

4.4 EER

EER is another extension to the E-R model proposed by Mani in [18]. The author extends the basic features of the E-R model with the following additional features:

- *order* in binary relationship types,
- *element-subelement* relationship types

Only binary relationship types are considered. Order in binary relationship types is proposed in the similar way as for the EReX model in [16] (but restricted only to the binary relationship types). The element-sublelement relationship types is a kind of relationship types with the label "has". One of the entity types has the cardinality (1,1) in each element-subelement relationship type. Attributes of the element-subelement relationship types are not considered. The concepts for modeling irregular and heterogeneous data and for modeling document-centric data is not considered.

The author proposes an algorithm for the translation of EER schemata to the XGrammar representation. The similar algorithm as in [16] is used, where the element-subelement relationship types are directly represented with nesting in XML.

4.5 XER

XER is an extension to the E-R model proposed by Sengupta et al. in [23]. The authors start from the properties of the XML Schema language and propose the following extensions to the E-R model:

- *ordered*, *unordered*, and *mixed* entity types
- *multivalued* attributes

The XER model is bounded by XML Schema. It is possible to model document centric data by the mixed entity types and it is possible to model ordering on the content of an entity type. However, only the binary relationship types without attributes and with the cardinality type $1 : N$ can be used. The authors do not introduce concepts for modeling irregular and heterogeneous structure.

The authors propose the extending constructs only in the form of examples and short descriptions. There is no formal background in their paper. However, it is possible to formally define the constructs following the formalism used in this paper.

4.5.1 Formal Description

Attributes of entity types in XER can be defined as multivalued and a number of values of a given attribute can be restricted by a cardinality constraint.

Definition 4.21 (Multivalued attributes) :
Given a data schema $DD = (U, D, dom)$ the set U_{xer} of *simple and multivalued attributes with a cadinality constraints* is defined as follows:

- $U \subseteq U_{xer}$
- If $X \in U$ and $m, n \in \{*\} \cup \{0, 1, \ldots\}$ then
$$\{X\}[m, n] \in U_{xer}$$
is a multivalued attribute with a cardinality constraint named X.

The semantics of the introduced kinds of attributes is given by the following definition. The definition extends the function *dom* to the function Dom_{xer} defined on U_{xer}.

Definition 4.22 (Domain function extension to U_{xer}) :
The $dom : U \to D$ function is extended to the $Dom_{xer} : U_{xer} \to D$ as follows:

- $\forall\, A \in U : \, Dom_{xer}(A) = dom(A)$
- For $\{X\}[m, n] \in U_{xer}$, $Dom_{xer}(\{X\}[m, n]) = \mathcal{P}^n_m(dom(X))$ where $\mathcal{P}^n_m(M) = \{M' \subseteq M : m \leq |M'| \leq n\}$.

The multivalued attributes with a cardinality constraint are a special case of the set-valued complex attributes.

There are ordered and unordered entity types in the XER model. We must specify an order between the attributes of an ordered XER entity type E and an order between the relationship types with a participant E. This order is specified directly in the definition of E.

Definition 4.23 (Ordered XER entity type) :
An *ordered XER entity type* has the form
$$E = (attr(E), rel(E), id(E))$$
where E is the name of the entity type, $attr(E)$ is an ordered list of attributes from U_{xer}, $rel(E)$ is an ordered list of the relationship types with a participant E and $id(E)$ is a non-empty subset of $attr(E)$ called the *key* of the entity type.

XER entity types are displayed by boxes with a title area showing the name of the entity type and the body showing the attributes and relationship types connected with the entity type in the prescribed order.

An ordered XER entity type E has an ordered list of attributes and an ordered list of relationship types. The extension of the entity type E is a set of tuples over a set of attributes from the ordered set $attr(E)$. A set of relationships with given participant from E^C must be ordered in the order prescribed by $rel(E)$.

An unordered XER entity type E can have only simple attributes and can participate in a relationship type with a maximal cardinality equal to 1. This condition arises from the XML Schema construct *all*. In a graphical representation a title area is labeled by ? symbol.

A mixed XER entity type E is defined in the same way as ordered XER entity type. Moreover, tuples from the extension E^C are marked as mixed. It means that in the XML representation of the given mixed tuple, the data of the tuple are mixed with a text. In a graphical representation a rounded rectangle is used.

The authors speak about binary relationship types with all types of cardinality constraints. However, only the binary relationship types with the cardinality constraint type $((1,1), N)$ are demonstrated, where it is clear how to represent them using nesting in XML data. It is not clear, how the relationship types with the other types of cardinality constraints would be expressed in XML schema languages.

Figure 4.7 displays a XER schema with departments modeled by the ordered entity type *Department* and professors in the departments modeled by the ordered entity type $Professor$. Each professor is a member of one department. The schema represents the papers written by a professor by the ordered entity type *Paper*. Each paper is written by one professor. A paper contains sections modeled by the ordered and mixed entity type *Section*. Each section is composed of a text mixed with emphasized texts and citations represented by the entity types *Emph* and *Cite*. The last two entity types are modeled as mixed because their instances contain only a text. It would be useful to have the constructs for modeling the irregular structure for modeling the content of the entity type *Section*. The group *CEmph* and *CCite* should be modeled as the repetition of the group with optional *CEmph* and *CCite*. It would allow the *Section* instance to contain the text mixed with an arbitrary sequence of *Emph* and *Cite* instances.

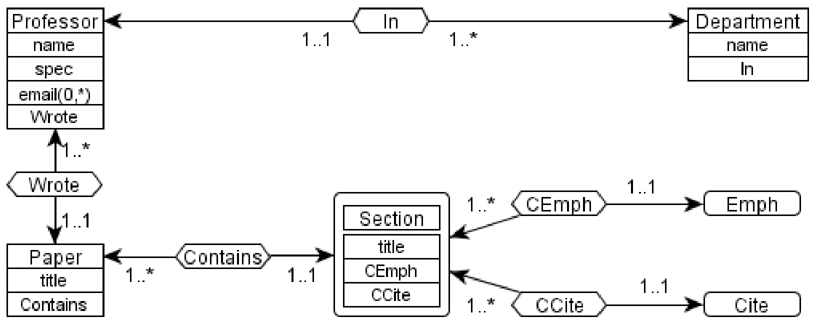

Figure 4.7: XER Diagram

4.5.2 Translation to XML Schema Languages

The authors give some examples of how to express XER modeling constructs in XML Schema. They do not introduce an algorithm. Ordered, unordered, and mixed entity types, and binary relationship types with the cardinality constraint type $((1,1),N)$ are directly represented by XML Schema constructs. It is not clear how binary relationship types with the other cardinality constraint types are translated to XML Schema. IS-A relationship types are represented using *choice* construct of XML Schema. A choice is created between the representation of the special entity types and is nested in the representation of the general entity type.

4.6 ERX

ERX is an extension to the E-R model proposed by Psaila in [22]. The author proposes the following extensions to the E-R model:

- *relationship types with alternatives*
- *containment relationship types*
- *unique attributes*
- *order attributes*
- *interfaces*

The author describes the introduced concepts formally. However, the algorithm for the translation of ERX schemata to XML schema languages is not proposed in the paper. Irregular and heterogeneous data can be modeled by the relationship types with alternatives. The author introduces the concept of containment relationship types (with arbitrary cardinalities) for modeling the hierarchical structure explicitly. Non-hierarchical relationship types can be used too. However, only the binary relationship types without attributes are allowed. Ordering on the extensions of entity types can be explicitly modeled by the order attributes with values from the set of natural numbers $\{1,2,\ldots\}$. The difference from the other conceptual models for XML is the concept of interfaces allowing the integration of different schemata. However, the author does not introduce any concepts for modeling document-centric data.

4.6.1 Formal Description

Entity types are comprehended as descriptions of complex (structured) concepts of the source XML documents. Entities of an entity type are comprehended as a particular occurences of a concept in a source documents. The author uses weak entity types for modeling concepts defined in the context of another concept. It is useful when modeling document-centric documents. For example, a chapter is a concept defined in the context of a book. Weak entity types are displayed using different graphical notation than the one described in Section 4.1. In the ERX graphical notation, relationship types connect a weak entity type with the entity types it depends on and special edges connect the partial key of the weak entity type with the relationship types as shown in Figure 4.8.

The author introduces a new kind of relationship types called *relationship types with alternatives* defined in the following definition. A relationship type with alternatives connects an entity type E with alternative entity types $E_1,\ldots,E_n$.

Definition 4.24 (Relationship type with alternatives) :
A *relationship type with alternatives* has the form

$$R = (E, (E_1, \ldots, E_n), (c, c_1, \ldots, c_n))$$

where R is the name of the relationship type, $E, E_1, \ldots, E_n$ are entity types and $c, c_1, \ldots, c_n$ are cardinality constraints. The entity types $E_1, \ldots, E_n$ are called the *alternatives.*

A relationship type with alternatives R specifies that an entity of the entity type E is connected with entities of the alternatives $E_1, \ldots, E_n$ by R. The cardinality constraint c for E considers all variants $E_1, \ldots, E_n$. It specifies the number of entities of alternatives $E_1, \ldots, E_n$ connected with an entity of E by R. Each alternative E_i has its own cardinality constraint specifying the number of entities of E connected with an entity of E_i by R.

The author introduces the concept of *containment relationship types.* Given two entity types E_1 and E_2, a containment relationship type leading from E_1 to E_2 denotes that in the source XML document each entity of E_1 contains entities of E_2. It is not a special kind of relationship types. Each binary relationship type can be denoted as a containment relationship type. In a graphical representation it is displayed by a dashed line connecting the relationship type with the enity type E_1.

The ERX model allows to model attributes of entity types. Attributes of relationship types are not considered by the author. Each attribute is required or optional. It is similar to the concept of optional attributes from the previous models. The author proposes the concept of *order attribute* denoting the order with which the entity of the entity type appears in the document. Values of an order attribute are from the set of natural numbers $\{1, 2, \ldots\}$. An attribute can be marked as *unique.* Each key is marked as unique imlicitly. It adds an integrity constraint meaining that only one entity of the entity type can have a given value for the attribute. Attributes are displayed as circles connected to their entity types by a solid line. A name of an attribute is displayed at the circle with R,I,O,U in parenthesis denoting if the attribute is R-required, I-implied(i.e. optional), O-order or U-unique.

The last feature considered by ERX is the concept of *interface.* An interface divides a schema to two parts that are semantically different. It is connected through a relationship types with one or more entity types in each of the two part. It is not an entity type, but a placeholder for entity types in the two parts.

Figure 4.8 displays an ERX schema representing professors, and books and papers written by the professors. There is the relationship type $AuthorOf$ connecting $Professor$ with alternatives *Paper* and *Book.* It means that each professor is an author of one or more papers and books. Each paper or book has one or more authors. Books contain chapters and chapters contain sections. Papers contain directly sections. It is represented by the containment relationship types *ContainsS* and *ContainsCh.* The relationship type *ContainsS* is the relationship type with alternatives *Paper* and *Chapter*. It means that both papers and book chapters contain sections. The containment relationship type *Contains* with alternatives *Emph* and *Cite* means that sections contain emphasized texts and citations. The ERX model does not offer mixed entity types which would be useful to model sections containing citations and emphasized texts mixed with a simple text. *Chapter*, *Section*, *Cite*, and *Emph* are weak entity types and they are ordered by the *order* attributes. There are the interfaces *IBook* and *IPaper* making the *Book* and *Paper*, respectively available outside the schema.

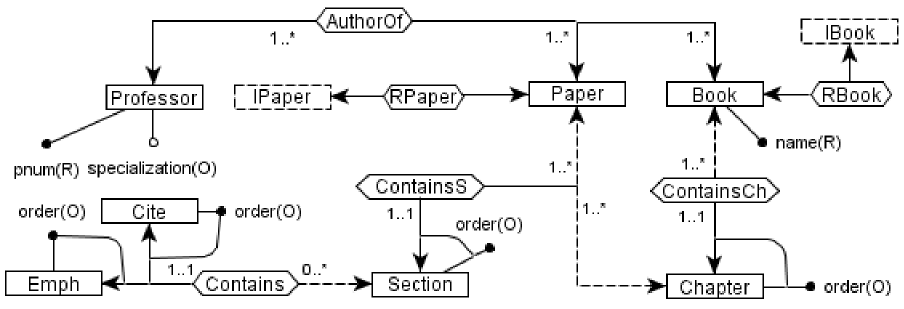

Figure 4.8: ERX Diagram

4.7 C-XML

The conceptual model called C-XML was proposed by Embley et al. in [9]. The authors propose an algorithm for the translation of the C-XML model to the XML Schema model and vice versa and they also study how the model can incorporate XQuery and how the model can be used for the data integration.

The authors state that the C-XML model has the same expressive power as XML Schema. It means that C-XML can represent each component and constraint in XML Schema and vice versa. C-XML is not described formally.

The basic modeling constructs of C-XML are *object sets*, *relationship sets*, and *constraints* over them. The concept of object sets is similar to the concept of entity types in the E-R model and the concept of relationship sets is similar to the concept of relationship types in the E-R model.

The concept of *lexical* and *nonlexical* object sets is introduced. Nonlexical object sets are the same as entity types of the E-R model. They have attributes and a key. Lexical object sets are used for modeling attributes of nonlexical object sets.

Lexical object sets are displayed as dashed boxes. The graphical notation for the other C-XML modeling constructs is similar to the graphical notation for the E-R modeling constructs. There are little differences in displaying keys and cardinality constraints.

The ordering between object sets connected with a non-lexical object set can be specified explicitly. For each connected object set A the previous object set B can be specified by marking A with $> B$.

The hierarchical structure can not be modeled explicitly in C-XML. There are no constructs for modeling document-centric data. No special concepts for modeling irregular and heterogeneous data are introduced.

4.7.1 Translation to XML Schema Languages

The authors introduce an algorithm for the translation of a C-XML schema C to a forest of schema trees F_C. The algorithm guarantees that F_C has a minimal number of schema trees, and that XML documents conforming to F_C have no redundant data with respect to constraints in C. The authors describe the algorithm in detail in [9] and in [10].

5 Hierarchical Conceptual Models for XML

The extensions of the E-R model allow to model conceptual schemata with a graph structure. However, XML schema languages allow to express relationship types only by nesting and references. It is possible to express all the relationship types from an E-R schema by references, but it leads to flat schemata and the advantages of the hierarchical structure of XML are not utilized. On the other hand, if the hierarchical structure is used to express relationship types in a conceptual schema the problem with the decision about what to nest arises. Another problem is how to represent n-ary relationship types and attributes of relationship types.

The problem is that there is no explicit information about a required nesting. Hierarchical conceptual models solve this problem by a special kind of *nesting binary relationship types*. These nesting binary relationship types describe the hierarchical structure explicitly. The nesting binary relationship type is oriented. The entity type the relationship type goes from is called the *parent participant* and the entity type the relationship type comes in is called the *child participant* of the relationship type. We say that the child participant is *nested* in the parent participant (by the nesting relationship type between them). Each entity type in a hierarchical schema can be a child participant of 0 or 1 nesting relationship types but not more. If a hierarchical model proposes another kinds of relationship types it must be clear how to express them by nesting or they must be used for modeling references explicitly.

In this section we describe a basic hierarchical conceptual model for XML first. In the next subsections, we describe conceptual models for XML based on the hierarchical approach. These models are: X-Entity [15], ORA-SS [7], and Semantic Networks for XML [11].

5.1 Basic hierarchical conceptual model for XML

The basic hierarchical conceptual model for XML can be easily defined as a restriction of the E-R model where only the binary relationship types with cardinality types $(1,1):1$ or $(1,1):N$ and without attributes are allowed. Each relationship type is oriented from the entity type with the arbitrary cardinality to the entity type with the cardinality $(1,1)$. This kind of relationship types may be called *nesting binary relationship types*. When modeling XML data, the nesting binary relationship types are represented by a nesting of elements on the XML logical level. They express a hierarchical structure on the XML logical level explicitly on the conceptual level. However, the semantics of nesting relationship types do not have to be only "part-of". It may be a general association too.

Such restrictions are too strong and do not allow to model conceptual schemata with richer semantics. Nor n-ary relationship types, nor attributes of relationship types can be modeled. Moreover, lots of redundancies may appear in schemata. There are some approaches extending this basic hierarchical model described in the following subsections.

5.2 X-Entity

X-Entity is a hierarchical model proposed by Loscio et al. in [15]. Probably, it was created as an extension of the E-R model, but the restrictions on relationship types proposed by the authors put the model to the class of hierarchical models. The authors propose the following modeling constructs:

- *cardinality of attributes*
- a kind of relationship types called *containment relationship types*
- a kind of structural constraints called *disjunction constraints*

The authors describe the extending modeling constructs formally. The model allows to model the hierarchical structure explicitly by the containment relationship types. It is possible to model irregular and heterogeneous structure by the disjunction constraints. However, it is not possible to model ordering and document-centric data. Other than the containment relationship types can not be used.

5.2.1 Formal Description

An entity type in a X-Entity schema does not represent general entities but it represents directly elements with a complex structure in an XML instance of the schema. The authors use the concept of the multivalued attributes with cardinality constraints as it was defined in the description of XER. The set of single and multivalued attributes with cardinality constraints was denoted by U_{xer}.

The X-Entity model allows to model only the containment binary relationship types without attributes. The containment relationship types represent a nesting between entities of related entity types. They are displayed as common relationship types.

Definition 5.1 (Containment relationship type) :
A *containment relationship type* has the form

$$R = (E_1, E_2, (min, max))$$

where R is the name of the relationship type, E_1 is the *parent entity type*, E_2 is the *child entity type*, and (min, max) defines the minimum and the maximum number of instances of E_2 that can be associated with an instance of E_1.

The authors propose a kind of integrity constraints called *disjunction constraints*. It is defined by the following definition.

Definition 5.2 (Disjunction constraint) :
A *disjunction constraint* has the form

$$D(E, \{d_1, \dots, d_n\})$$

where E is an entity type and d_i is an attribute or a set of attributes of the entity type E or a containment relationship type or a set of containment relationship types with the parent entity type E.

A disjunction constraint D specifies that each entity of E can be associated with only one of the concepts specified in the disjunction constraint. It is displayed as an arc, which traverses the attributes or the relationship types participating in the constraint definition. If d_i denotes a set of attributes or a set of containment relationship types then the attributes/relationship types are attached to a single line connecting them to the entity type. Disjunction constraints are used to represent choice constraints which can be specified by XML schema languages.

Figure 5.1 displays an X-Entity schema representing departments, professors in the departments, and courses teached by the professors. Each course is teached by one professor. Each professor has specified his name as one value or as the first name and the second name. The schema is formally described as follows.

$$\begin{aligned}
Department &= (\{name, \{phone\}[1,*]\}, \{name\}),\\
Professor &= (\{pnum, name, first, second\}, \{pnum\}),\\
Course &= (\{code, name\}, \{code\}),\\
In &= ((Department, Professor), (1,*)),\\
Teaches &= ((Professor, Course), (1,*)),
\end{aligned}$$

$$D(Professor, \{name, \{first, second\}\})$$

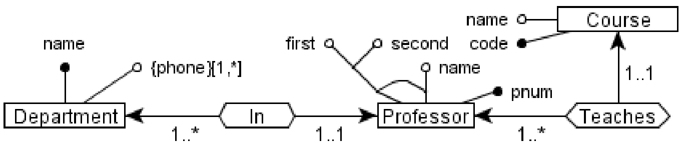

Figure 5.1: X-Entity Diagram

5.2.2 Translation to XML Schema Languages

The authors propose an algorithm for the translation from XML Schema to X-Entity. An inverse algorithm is not proposed. However, the modeling constructs of the X-Entity model can be translated to XML Schema directly because only the containment relationship types are used.

5.3 ORA-SS

ORA-SS is a rich hierarchical conceptual model for XML proposed by Dobbie et al. in [7]. They offer the following extending features:

- *cardinality constraints* for the both participants of hierarchical relationship types

- *degree* of hierarchical relationship types
- *attributes of relationship types*
- *ordering*
- *disjunction*
- *references*

ORA-SS has three basic modeling constructs: object types, relationship types, and attributes. The object type construct is similar to the entity type from the E-R model. Relationship types between object types represent hierarchical relationships. Non-hierarchical relationships can be modeled by the references. The authors introduce the concept of n-ary hierarchical relationship types and attributes of hierarchial relationship types. All the types of ordering can be modeled. Irregular and heterogeneous structure can be modeled by the disjuntion. However, there are no constructs for modeling document-centric data.

The authors do not propose the formalism for the ORA-SS model, but it can be described following the formalism for the E-R model.

Applications of ORA-SS are introduced in [8], [2], [3], [4], and [26]. These papers use the ORA-SS model for the normalization of conceptual schemata, the conceptual modeling of hierarchical views, the data translation between conceptual hierarchical views and the integration of different conceptual schemata to an overall schema.

5.3.1 Formal Description

There are object types instead of entity types in the ORA-SS model. The ordering between relationship types leading from an object type is specified. Each object type is placed in the hierarchical structure of a schema by a nesting relationship type coming to it. The authors propose the notion of attributes of nesting relationship types. The problem is that a nesting relationship type is expressed by a nesting in an XML schema language and can not have its attributes assigned directly. The attributes of the nesting relationship type must be assigned to the child participant of the relationship type or to some of its descendants.

Attributes ORA-SS allows to model composite attributes, unordered/ordered multivalued attributes and disjunctive attributes. Moreover, the authors propose the concept of derived attributes and attributes with an unspecified heterogeneous structure denoted by ANY. Derived attributes are not formalized in the following definition of ORA-SS attributes. Each attribute can be marked as derived.

Definition 5.3 (ORA-SS attributes) :
Given a data schema $DD = (U, D, dom)$ the set U_{ora} of ORA-SS attributes is defined as follows:

- $U \subseteq U_{ora}$

- $ANY \in U_{ora}$
- If $X_1, \ldots, X_n \in U_{ora}$ (not necesarilly distinct) and X is an attribute name then
$$X(X_1, \ldots, X_n) \in U_{ora}$$
is a *composite attribute* named X.
- If $X \in U_{ora}$ and $m, n \in \{*\} \cup \{0, 1, \ldots\}$ then
$$\{X\}[m, n] \in U_{ora}$$
is an *unordered multivalued attribute.*
- If $X \in U_{ora}$ and $m, n \in \{*\} \cup \{0, 1, \ldots\}$ then
$$\langle X\rangle[m, n] \in U_{ora}$$
is an *ordered multivalued attribute.*
- If $X_1, \ldots, X_n \in U_{ora}$ and X is an attribute name then
$$X(X_1|\ldots|X_n) \in U_{ora}$$
is a *disjunctive attribute* named X.

The semantics of the introduced kinds of attributes is given by the following definition. The definition extends the function *dom* to the function Dom_{ora} defined on U_{ora}.

Definition 5.4 (Domain function extension to U_{ora}) :
The $dom : U \to D$ function is extended to the $Dom_{ora} : U_{ora} \to D$ function as follows:

- $Dom_{ora}(ANY)$ is a set of all possible attribute values.
- $\forall A \in U : Dom_{ora}(A) = dom(A)$.
- For $X(X_1, \ldots, X_n) \in U_{ora}$:
$$Dom_{ora}(X) = Dom_{ora}(X_1) \times \ldots \times Dom_{ora}(X_n)$$
- For $\{X\}[m, n] \in U_{ora}$:
$$Dom_{ora}(X) = \mathcal{P}^n_m(Dom(X))$$
where $\mathcal{P}^n_m(M) = \{M' \subseteq M : m \leq |M'| \leq n\}$.
- For $\langle X\rangle[m, n] \in U_{ora}$:
$$Dom_{ora}(X) = \\ \{\langle x_1, \ldots, x_k\rangle : m \leq k \leq n \wedge (\forall 1 \leq i \leq k)(x_i \in Dom_{ora}(X))\}$$
where $\langle x_1, \ldots, x_k\rangle$ denotes an ordered list of items (not necessarily distinct).

- For $X(X_1|\ldots|X_n) \in U_{ora}$:

$$Dom_{ora}(X) = Dom_{ora}(X_1) \cup \ldots \cup Dom_{ora}(X_n)$$

The composite attributes are equivalent to the tuple-valued attributes. The unordered multivalued attributes are the anonymous set-valued attributes and the ordered multivalued attributes are the anonymous list-valued attributes. However, a named multivalued attribute can be specified as a composite attribute with one multivalued component. The disjunctive attributes are equivalent to the variant-valued attributes.

Object types ORA-SS object types are defined by the following definition.

Definition 5.5 (ORA-SS object type) :
An *ORA-SS object type* has the form

$$E = (attr(E), hier(E), id(E))$$

where

- E is the name of the object type
- $attr(E)$ is an ordered list of attributes from U_{ora} or 2-tuples (A, l), where A is an attribute from U_{ora} and l is the name of an ancestor relationship type of the object type; if the member of the list is an attribute it is an attribute of the object type; if the member of the list is a 2-tuple (A, l) the attribute A is an attribute of the ancestor relationship type with the name l
- $hier(E)$ is an ordered list of ORA-SS relationship types with the parent participant E
- $id(E)$ is a non-empty subset of the attributes (not the attributes from 2-tuples) from $attr(E)$ called the *key* of the object type.

The result of the definition of the ORA-SS object types is that an object type has assigned the attributes of ancestor relationship types. On the conceptual level, they are the attributes of the relationship types. However, on the XML logical level, they are undistinguishable from the attributes of the object type.

Figure 5.2 displays an ORA-SS schema modeling projects and professors by the object types $Project$ and $Professor$, respectively. Professors are members of projects. It is represented by the relationship type $Member$. A professor may be a member of zero or more projects and a project may have one or more members. We need a list of projects and for each project a list of its member professors. Moreover, for each professor in the project a date when the professor became a member of the project must be stored. It is represented by the attribute *since* of the relationship type $Member$. However, the attribute can not be assigned directly to the relationship type. It must be assigned to the object type $Professor$ and connected with the relationship type by the label $Member$ as shown in the figure. The schema is formally described as follows.

$$\begin{aligned}Project &= ((projno, title), (Member), \{projno\}),\\ Member &= (Project, Professor, (0,*), (1,*)),\\ Professor &= ((profno, name, (since, Member)), (), \{profno\})\end{aligned}$$

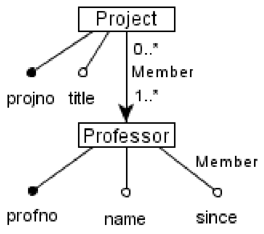

Figure 5.2: ORA-SS Diagram - ORA-SS Object Types

The authors do not define the extension of an object type. However, following the previous formalism the extension E^C of an object type E can be defined in the same way as the extension of an entity type, i.e. it is a set of tuples on the attributes from $attr(E)$ (the attributes in 2-tuples are considered too). The tuples from E^C are called *objects* of the object type E.

The previous definition of object types allows an object type to have only one key. This key is a primary key. However, the authors propose object types with the primary key and zero or more candidate keys. It is not formalized by the definition. A candidate key introduces a similar integrity constraint as the primary key.

Nesting relationship types The authors of the ORA-SS model propose more advanced kind of nesting relationship types than the ones described in Subsection 5.1. The ORA-SS relationship types are nesting binary relationship types connecting two object types or another ORA-SS relationship type with an object type (which is always the child participant). Moreover, the cardinality constraints for the both participants can be specified. Following the formalism used in previous definitions of different kinds of relationship types, the ORA-SS relationship types can be defined as follows.

Definition 5.6 (ORA-SS relationship type) :
An *ORA-SS relationship type* has the form

$$R_2 = (R_1, E_2, card_p, card_{ch})$$

where R_2 is the name of the ORA-SS relationship type, R_1 is an object type or another ORA-SS relationship type, and E_2 is an object type. E_2 is called the *child participant* in R_2. If R_1 is an object type it is called the *parent participant* in R_2. Otherwise, the child participant in R_1 is called the *parent participant* in R_2. The last two members $card_p$ and $card_{ch}$ are the cardinality constraints for the parent participant and for the child participant.

Each ORA-SS relationship type goes from its parent participant to its child participant. Let $R_2 = (E_1, E_2)$ be an ORA-SS relationship type. It goes from the object type E_1 to the object type E_2. It represents the nesting of the object type E_2 in the object type E_1, i.e. the objects of the object type E_2 are nested in the objects of the object type E_1 by relationships from the relationship type R_2.

Now, let $R_3 = (R_2, E_3)$ be another ORA-SS relationship type. It goes from the relationship type R_2 to the object type E_3. It represents the nesting of the object type E_3 in the object type E_2 nested in the object type E_1.

Let e_1 be an object of the object type E_1. For each relationship r_2 of the relationship type R_2 going from e_1 to an object e_2 of the object type E_2 the object e_2 is nested in the object e_1. If the object e_1 is considered the objects of the object type E_2 nested in e_1 by the relationship type R_2 can be considered in every situation.

Now, let e_2 be an object of the object type E_2. There is the relationship type R_3 nesting the objects of the object type E_3 to e_2. However, an object e_1 of the object type E_1 such that e_2 is nested in e_1 by R_2 is needed. If only e_2 is considered no objects of E_3 nested in e_2 can be considered because it has no meaning. If e_2 nested in e_1 by R_2 is considered the objects of the object type E_3 nested in e_2 by R_3 can be considered. If e_2 is nested in e'_1 by R_2, where $e_1 \neq e'_1$, the objects of the object type E_3 nested in e_2 by R_3 can be considered but they can be different from the previous case.

Degree of an ORA-SS relationship type An ORA-SS relationship type relates two or more object types. The number of related object types is called the *degree* of the relationship type. It is defined by the following definition.

Definition 5.7 (Degree of an ORA-SS relationship type) :
Let $R_2 = (R_1, E_2)$ be an ORA-SS relationship type. If R_1 is an object type then the degree of R_2, denoted by $degree(R_2)$, is 2. If R_1 is an ORA-SS relationship type then the degree of R_2 is $degree(R_1) + 1$.

The following definition defines the *context* of an object. If an object e nested by a relationship type R with $degree(R) = n$ is considered its context contains the objects that must be considered too.

Definition 5.8 (Context of an ORA-SS object) :
Let $E_1, \dots, E_n$ be ORA-SS object types and $R_2 = (E_1, E_2), R_i = (R_{i-1}, E_i), 3 \leq i \leq n$ be ORA-SS relationship types. Let e_j be an object of the object type E_j for $1 \leq j \leq n$ such that e_k is nested in e_{k-1} for $2 \leq k \leq n$. An ordered list of the objects $(e_1, \dots, e_{n-1})$ is called the *context* of the object e_n.

If e is an object nested by a relationship type R with $degree(R) = 2$, the context of e contains only one object. It means that the object e nested by R can be considered if its parent is considered too. If $degree(R) = n$, the context of e contains $n - 1$ objects. All of these objects must be considered when the object e nested by R is considered.

Cardinality of an ORA-SS relationship type Consider again the relationship types R_2 and R_3. Let $card^2_p$ and $card^2_{ch}$ be the parent participant and the child participant cardinality constraints for R_2 and $card_p^3$ and $card_{ch}^3$ be the parent participant and the child participant cardinality constraints for R_3. There is a difference between the semantics of both cardinalities, because of the different degrees of R_2 and R_3.

The cardinality constraints for R_2 have the classical meaning because $degree(R_2) = 2$. The cardinality constraint $card_{ch}^2$ constraints the number of objects of the object type E_2 that can be nested in a given object e_1 of the object type E_1. Similarly, the cardinality constriant $card_p^2$ constraints the number of objects of the object type E_1 a given object e_2 of the object type E_2 can be nested in.

The cardinality constraints for R_3 do not constraint a nesting of E_3 in E_2, because $degree(R_3) = 3$. If e_1 and e_2 are objects of the object types E_1 and E_2, respectively, the cardinality constraint $card_{ch}^3$ constraints the number of objects of the object type E_2 with the context (e_1, e_2). Similarly, the cardinality constraint $card_p^3$ constraints the number of contexts of a given object of the object type E_3 nested by R_3.

A relationship type is displayed as an arrow going from the parent participant to the child participant. The parent cardinality constraint is displayed at the parent participant side of the arrow and the child cardinality constraint is displayed at the child participant side of the arrow. The arrow is labeled by the degree of the relationship type and by the name of the relationship type.

Figure 5.3 displays two ORA-SS schemata. There are projects, professors, and papers represented in the both schemata. In the both schemata, there is the object type $Professor$ nested in the object type $Project$ and the object type $Paper$ nested in $Professor$. The relationship types state that professors are members of the projects and they write papers. Each paper is written by one or more professors and each professor is a member of zero or more projects. The relationship type $AuthorOf$ has the attribute $pages$ nested in $Paper$ representing the number of pages written by a professor in a paper.

The difference between the two schemata is that the schema on the left hand side has the relationship type $AuthorOf$ with the degree 2 and the schema on the right hand side has the relationship type $AuthorOf$ with the degree 3. The relationship type $AuthorOf$ in the left hand side schema represents the papers written by a professor. All of the papers are repeated in each project the professor is a member of. Moreover, it can not be distinguished which papers the professor wrote in a given project. The relationship type $AuthorOf$ in the right hand side schema represents the papers written by a professor being a member of a project. If a professor is nested in a project, only the papers written by the professor during his work in the project are nested. The schemata are formally described as follows.

$$\begin{aligned}
Project &= ((projno, title), (Member), \{projno\}),\\
Professor &= ((profno, name), (AuthorOf), \{profno\}),\\
Paper &= ((code, title, (pages, AuthorOf)), \{code\}),\\
Member &= (Project, Professor, (0, *), (1, *)),
\end{aligned}$$

For the left hand side schema:

$$AuthorOf = (Professor, Paper, (1, *), (0, *)),$$

And for the right hand side schema:

$$AuthorOf = (Member, Paper, (1, *), (0, *)),$$

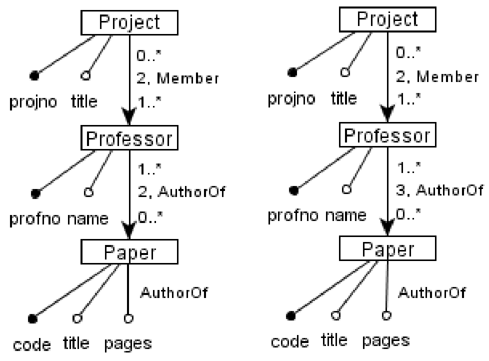

Figure 5.3: ORA-SS Diagram - Projects

Ordering The authors of the ORA-SS model distinguish 3 kinds of ordering.

- the values of an attribute can be ordered
- the attributes of an object type and relationship types going from it can be ordered
- the objects nested by a relationship type can be ordered

The ordering of the values of an attribute was defined in the definition of the ordered multivalued attributes. If an object type

$$E = ((A_1, \ldots, A_m), (R_1, \ldots, R_n))$$

is marked as ordered then the list of its attributes and relationship types going from it is ordered. The ordering is $A_1 \leq \cdots \leq A_m \leq R_1 \leq \cdots \leq R_n$ and specifies the ordering on the attributes and the nested objects of a given object of the object type E. This type of ordering is displayed by $<$ symbol at the object type's box.

The ordering on the attributes and the relationship types of the object type does not specify the ordering on a relationship type going from the object type. If R_1 and R_2 are two relationship types going from an ordered object type E then an object of E has nested relationships from R_1 first and after them has nested relationships from R_2. However this ordering does not specify that objects nested by R_1 or R_2 are ordered. It is specified by the ordering on relationship types. If R_1 nesting E_1 in E is specified as ordered then objects $e_1, \ldots, e_n$ of E_1 nested by R_1 in an object e of E are ordered. It is displayed by $<$ symbol at the relationship type's arrow.

Figure 5.4 displays an ORA-SS schema representing the structure of papers. Papers are represented by the ordered object type *Paper*, i.e. its attributes and nested object types are nested in the following order:

1) code
2) title
3) ordered list of authors
4) abstract
5) sections
6) bibliography

The relationship type *Ps* nesting *Section* in *Paper* is ordered. It means that the sections of a paper are ordered. The same is the relationship type *Psp* nesting *Paragraph* in *Section*.

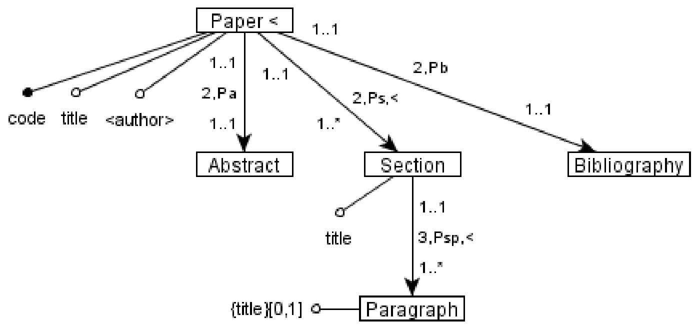

Figure 5.4: ORA-SS Diagram - Ordering in Papers

Disjunction The authors propose modeling constructs for disjunctive attributes and disjunctive object types. The disjunctive attributes were defined in the definition of attributes. The disjunctive object types are modeled by the disjunctive relationship types. Following the formalism used in this paper, the ORA-SS disjunctive relationship types can be defined as follows.

Definition 5.9 (ORA-SS disjunctive relationship type) :
An *ORA-SS disjunctive relationship type* has the form

$$R_2 = (R_1, (E_{2,1}, \ldots, E_{2,n}), card_p, card_{ch})$$

where R_2 is the name of the ORA-SS relationship type, R_1 is an object type or another ORA-SS relationship type, and $E_{2,1}, \ldots, E_{2,n}$ are object types. $E_{2,1}, \ldots, E_{2,n}$ are called the *child participants* in R_2. If R_1 is an object type it is called the *parent participant* in R_2. Otherwise, the child participant in R_1 is called the *parent participant* in R_2. The last two members $card_p$ and $card_{ch}$ are cardinality constraints for the parent participant and for the child participants, respectively.

A disjunctive relationship type is displayed by a diamond with the symbol '|' in the middle. The diamond is connected by a solid line with the parent participant and by an arrow going from the diamond to each of the child participants.

However, the authors do not define the extensions of the disjunctive relationship types. Because of this, the semantics of the disjunctive relationship types is not clear. Let $R_2 = (E_1, (E_{2,1}, \ldots, E_{2,n}))$ be an ORA-SS disjunctive relationship type (where $degree(R_2) = 2$). On the instance level, it is not clear whether an object e_1 of the object type E_1 can have nested objects of the all object types $E_{2,1}, \ldots, E_{2,n}$ or it can have nested objects of only one of the object types. The former corresponds to applying the disjunction on the instace level (for each relationship of the relationship type R_2 one of the child participants is instanciated) and the other corresponds to applying the disjunction on the schema level (just one of the nested object types is selected and its objects are nested by R_2).

Figure 5.5 displays an ORA-SS schema representing professors, and the books and papers they have written. With no more constraints, the schema has two meanings:

- Each professor writes only books or only papers.
- Each professor writes a mixture of books and papers.

The disjunction is formally described as follows.

$$AuthorOf = (Professor, (Book, Paper), (1, *), (0, *))$$

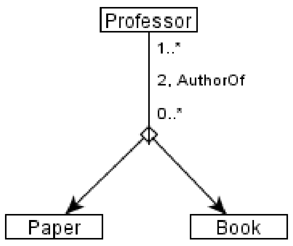

Figure 5.5: ORA-SS Diagram - Disjunction

References The authors propose another kind of relationship types called *references*. A reference is a relationship type between two object types. It is oriented from the *reference object type* to the *referenced object type*. However, the reference is not materialized by the nesting as in the case of the nesting relationship types. The authors do not specify (purposely) how the reference relationship types are materialized. They assume some reference mechanism on the instance level. References can be formalized by the following definition.

Definition 5.10 (ORA-SS reference relationship type) :
An *ORA-SS reference relationship type* has the form

$$R = (E_1, E_2)$$

where R is the name of the ORA-SS reference relationship type, E_1 is the *reference object type* and E_2 is the *referenced object type*. The relationship type R is oriented from E_1 to E_2.

Because of the hierarchical structure of the ORA-SS schemata, one real world object type can be represented by more conceptual object types in a schema. References allow to interconnect these object types. The division of real world object types to more conceptual object types is used for the normalization of schemata, for example. References are also used to model recursive and symmetric relationships.

Figure 5.6 displays an ORA-SS schema representing professors as members of departments and professors as participants in projects. The object type *Professor* nested in the object type *Project* has nested only *Paper* representing papers written by a professor during his work in the project. The information tied with a certain professor as his name and the address is represented in the object type *Professor* nested in the object type *Department*. *Professor* nested in *Project* is connected to *Professor* nested in *Department* by the reference relationship type. It avoids a redundancy in the instance level.

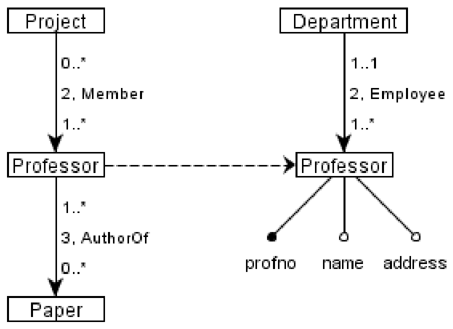

Figure 5.6: ORA-SS Diagram - Reference

5.3.2 Translation to XML Schema Languages

The algorithm for the translation ORA-SS schemata to DTD schemata is introduced in [8]. The ORA-SS model is a hierarchical conceptual model, so the translation to the XML schema languages is clear. However, the ORA-SS allows to model n-ary relationship types and attributes of relationship types. The semantics of this modeling constructs can not be modeled in XML schema languages if we want to maximize the utilization of the nesting. Hence, the part of the semantics described by an ORA-SS schema is lost during the translation.

The translation algorithm is clear. First, for each object type O an element type definition

$$<!ELEMENT \quad O \quad (SubelementList))>$$

is created. *SubelementList* contains the names of the elements created for the object types nested in O in the prescribed order. If there is a disjunction relationship type going from the object type O then the included object types are divided in *SubelementList* by | mark. Each simple attribute is translated as XML attribute. Each single valued simple attribute is replaced with its components and translated separately. The other types of attributes are translated as XML elements. N-ary relationship types are comprehended as binary relationship types during the translation. Attributes of relationship types are comprehended as attributes of the object types in which they are nested in the ORA-SS schema.

5.4 Semantic Networks for XML

The semantic network model for XML was introduced by Feng et al. in [11]. The model is a little extension to the basic hierarchical conceptual model described in Section 5.1. The authors propose some new kinds of relationship types and some new kinds of integrity constraints.

The authors introduce a formalism for the semantic network model for XML. The concepts of the model are different from the concepts of the well-known E-R model and the other models described in this paper. The formalism introduced by the authors is used here without modifications.

Only the binary relationship types without attributes can be modeled in the semantic network model. The model introduces other kinds of relationship types than the hierarchical relationship types. Hierarchical relationship types must have the parent participant cardinality equal to $(1, 1)$. Irregular and heterogeneous structure and ordering can be modeled. However, document-centric data can not be modeled.

5.4.1 Formal Description

A semantic network for XML is an oriented graph constisting of nodes connected by directed labeled edges. In addition, constraints can be defined over these nodes and edges. A semantic network model for XML consists of the four components:

- A set of nodes $\mathcal{N}_{ode}$, representing real-world objects;
- A set of directed edges $\mathcal{E}_{dge}$, representing semantic relationships between the objects;
- A set of labels $\mathcal{L}_{abel}$, denoting different types of semantic relationships;
- A set of constraints $\mathcal{C}_{onstraint}$, defined over the nodes and edges.

Nodes are not equivalent to the entity types from the E-R model. Nodes are categorized into *basic nodes* and *complex nodes*. The basic nodes are the leaf nodes in the semantic network diagram and the complex nodes are the internal nodes. The basic nodes are used to model simple objects and simple attributes and the complex nodes are used to model complex objects and complex attributes.

Nodes Each node has the content. The content of a basic node is called the *basic content*. The basic content can be an atomic content (string, integer, ...) or it can be described by the set, bag, list or variant-valued constructor defined earlier for the E-R model. The tuple constructor is not allowed in the semantic network model. Composite attributes must be modeled by the complex nodes. Hence, it is not possible to apply the set, bag, list or variant-valued constructor to the domain of a composite attribute in the semantic network model.

The content of a complex node is called the *complex content* and it refers to some other nodes through directed labeled edges. The complex content is defined by the following definitions. First, the authors propose the concepts of *connection*, *connection cluster*, and *connection cluster set.*

Definition 5.11 (Connection of a node) :
A *connection* of a node $n \in \mathcal{N}_{ode}$ is an ordered pair $< l, n' >$, where l is a label in $\mathcal{L}_{able}$ and n' is a node in $\mathcal{N}_{ode}$, representing that node n is connected to node n' via relation l.

Definition 5.12 (Connection cluster of a node) :
A *connection cluster* of a node $n \in \mathcal{N}_{ode}$ is an ordered pair $< l, ns >$, where l is a label in $\mathcal{L}_{able}$ and ns is a set of nodes in $\mathcal{N}_{ode}$, representing that node n is connected to each node in ns via relation l.

Definition 5.13 (Connection cluster set of a node) :
A *connection cluster set* of a node $n \in \mathcal{N}_{ode}$ is a set of connection clusters, $\{< l_1, ns_1 >, \ldots, < l_k, ns_k >\}$, where $\forall\, i\, \forall\, j\, (1 \leq i, j \leq k)\, (i \neq j \leftrightarrow l_i \neq l_j)$.

Definition 5.14 (Complex content of a complex node) :
A *complex content* of a complex node is a connection cluster set.

Now, nodes can be formally defined.

Definition 5.15 (Node) :
A *node* $n \in \mathcal{N}_{ode}$ is a 4-tuple $(n_{id}, n_{name}, n_{category}, n_{content})$, consisting of an unique node identifier n_{id}, a node name n_{name}, a node category $n_{category}$ indicating whether the node n is basic or complex, and a node content $n_{content}$.

Edges Edges connect nodes in the semantic network model. They are the similar concept to relationship types from the ORA-SS model. However, only the binary edges (connecting two nodes) without attributes are allowed in the semantic network model. Another difference is that different concepts are modeled using edges. These concepts are *generalization*, *aggregation*, *association*, and *of-property.* Each edge is labeled by g, a, s or p to denote if it is generalization, aggregation, association, or of-property, respectively.

A generalization is used to model $IS - A$ hierarchies between general and special concepts. An aggregation is used to model the nesting between concepts. An association is used to model relationships between real world objects on the same level (one can not be nested in the other). An of-property specifies the subsidiary attribute of an object (i.e. attributes are modeled as basic nodes connected to their nodes by of-property edges).

Definition 5.16 (Edge) :
An *edge* $e \in \mathcal{E}_{dge}$ is a triple $(e_{label}, e_{source\_node}, e_{target\_node})$, consisting of a label $e_{label} \in \mathcal{L}_{abel}$ stating the link type, the source node of the edge $e_{source\_node} \in \mathcal{N}_{ode}$, and the target node of the edge $e_{target\_node} \in \mathcal{N}_{ode}$.

Integrity constraints Different constraints can be specified in the semantic network for XML. Constraints can be specified over a node, over an edge, and over a set of edges.

Integrity constraints over a node are *uniqueness* specifying a uniqueness of the content of the node, *referential integrity* specifying that the content of the node is linked to the content of another node, and *domain constraint* constraining the content of the node by several rules as a minimal and a maximal length of a string content etc.

Integrity constraints over an edge comprehend *cardinality* constraints for the child nodes of the node. Further, there are some less traditional constraints called *homogeneous/heterogeneous composition* and *adhesion*. An aggregation edge can be constrained as homogeneous or heterogeneous composition. The former is used when a whole object is only made up of part objects of the same type. The later is as the opposite. The adhesion constraints indicate when one peer appears in a relationship, whether another peer must coexist. It can be used to model required/optional attributes, for example.

Integrity constraints over a set of edges comprehend *ordered composition* and *exclusive disjunction*. These constraints were described in the descriptions of the previous models.

Figure 5.7 displays a semantic network schema. There are departments represented by the complex node *Department* and professors in the departments represented by the complex node $Professor$. The content of the complex node $Professor$ is ordered. For each professor there are the papers he wrote represented by the complex node *Paper*. Each paper may be composed of chapters or sections, but not both (the exclusive constraint). The courses offered by a department are represented by the complex node $Course$. $Professor$ is associated with $Course$. It represents the relationships between a professor and the courses he teaches.

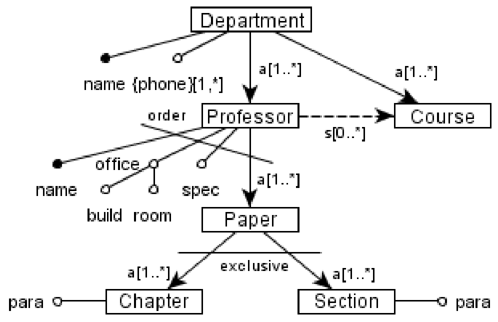

Figure 5.7: Semantic Network Diagram

5.4.2 Translation to XML Schema Languages

In [11], the authors describe possibilities of how to translate semantic network model to XML Schema language.

The basic nodes with atomic contents are translated using the XML Schema build-in datatypes and the XML Schema facets mechanism. If a list constructor was used then the XML Schema $< xsd : list >$ mechanism is used. If a variant constructor was used then the XML Schema $< xsd : union >$ mechanism is used. If a set and bag constructor was used, the basic node is translated as an element. To ensure the uniqueness of values in a set the $< xsd : unique >$ mechanism can be used.

The complex nodes are translated to complex types in the XML Schema language. The of-property edge is translated using $< xsd : attribute >$ concept. The aggregation edge is materialized by the nesting in XML Schema. The association edge can be materialized by the nesting in XML Schema if it is possible or by the ID/key mechanism in XML Schema. The generalization edge is translated using the derivation of complex types by the extension and the rescriction concepts in XML Schema.

The integrity constraints allowed by the semantic network model can be easilly translated to XML Schema using $< xsd : key >$, $< xsd : keyref >$, $< xsd : choice >$, and cardinality mechanisms.

6 Comparison of Described Conceptual Models

In this section, we compare the conceptual models described in this paper. The comparison is made against the general requirements and the modeling constructs requirements introduced in Section 2.

There are four comparative tables. Table 1 and Table 2 compare the models based on the E-R model. Table 3 and Table 4 compare the models based on the hierarchical approach. The E-R model and the basic hierarchical model are compared too.

We are not able to decide, which of the previous two approaches (E-R extensions, hierarchical modeling) is better for the conceptual modeling of XML data. Conceptual models based on the E-R model allow user to create a schema with no metadata redundancy, but there is the problem with the modeling of the specific XML features. Hierarchical conceptual models solve the problem with a hierarchical structure of XML, but there arises problems such as data and metadata redundancy, modeling of attributes of relationship types, and modeling of n-ary relationship types.

There are requirements that are not met by the described models. The modeling of document centric data and the reuse of content is problematic. The important requirement on the integration of conceptual schemata is solved only by the ORA-SS model. None of the models solve the problem of the integration with the semantic web technologies.

Table 6.1: Comparison of ER Based Models against the General Requirements

Property	ER	ER-B	EReX	EER	XER	ERX	C-XML
· independence on XML schema languages							
	√	–	√	√	–	√	√
· formal foundations							
	√	√	√	–	–	√	–
· graphical notation							
	√	√	√	√	√	√	√
· logical level mapping							
– relational model							
	√	√	–	–	–	–	–
– tree grammar based XML schema languages							
	–	√	√	√	√[1]	–	√
– pattern based XML schema languages							
	–	–	–	–	–	–	–
– utilization of hierarchical structure of XML							
	–	–	√	√	√	–	√
· different structures on the logical level							
– conceptual hierarchical views							
	–	–	–	–	–	–	–
– translation between hierarchical views							
	–	–	–	–	–	–	–
· semantic web mapping							
	–	–	–	–	–	–	–

[1] formal description is missing

Table 6.2: Comparison of ER Based Models against the Modeling Constructs Requirements

Property	ER	ER-B	EReX	EER	XER	ERX	C-XML
· hierarchical relationship types							
	–	–	–	√	–	√	–
– $M : N$ cardinality							
– N-ary							
– attributes							
	–	–	–	–	–	–	–
· non-hierarchical relationship types							
	√	√	√	√	√	√	√
– $M : N$ cardinality							
	√	√	√	√	–	√	√
– N-ary							
– attributes							
	√[1]	√	√	√	–	–	√
· ordering							
– on the values of an attribute							
	√[1]	–	–	–	–	–	–
– on the content of a concept							
	–	–	–	–	√[2]	–	√
– on the participant of a relationship type							
	–	–	√	√	–	√[3]	–
· irregular and heterogeneous structure							
– variant-valued attribute constructor							
	√[1]	√	–	–	–	–	–
– disjunctive constraints on relationship types							
	–	√[4]	√[4]	–	–	√	–
· document-centric data							
– basic mixed content							
	–	–	–	–	√	–	–
– generalized mixed content							
	–	–	–	–	–	–	–
· reuse of content							
– IS-A or the category concept							
	√	√	√	–	√	√	√
– named types and groups of concepts							
	–	–	–	–	–	–	–
· integration of conceptual schemata							
– modeling constructs							
	–	–	–	–	–	√	–
– algorithms for merging schemata							
	–	–	–	–	–	–	–
– algorithms for the data translation between schemata							
	–	–	–	–	–	–	–

[1] with the complex attributes extension
[2] unordered content in restricted by the restrictions of xsd:all construction in XML Schema
[3] only by the concept of ordered attributes, native XML ordering is not utilized
[4] using the category concept

Table 6.3: Comparison of Hierarchical Models against the General Requirements

Property	Hier	X-Entity	ORA-SS	Sem.net.
· independence on XML schema languages				
	✓	✓	✓	✓
· formal foundations				
	✓	✓	✓	✓
· graphical notation				
	✓	✓	✓	✓
· logical level mapping				
– relational model				
	–	–	–	–
– tree grammar based XML schema languages				
	✓	✓	✓	✓
– pattern based XML schema languages				
	–	–	–	–
– utilization of hierarchical structure of XML				
	✓	✓	✓	✓
· different structures on the logical level				
– conceptual hierarchical views				
	–	–	✓	–
– translation between hierarchical views				
	–	–	✓	–
· semantic web mapping				
	–	–	–	–

Table 6.4: Comparison of Hierarchical Models against the Modeling Constructs Requirements

Property	Hier	X-Entity	ORA-SS	Sem.net.
· hierarchical relationship types				
	√	√	√	√
– $M : N$ cardinality – N-ary – attributes				
	–	–	√	–
· non-hierarchical relationship types				
	–	–	√	√
– $M : N$ cardinality – N-ary – attributes				
	–	–	–[1]	–
· ordering				
– on the values of an attribute				
	√	–	√	√
– on the content of a concept				
	–	–	√	√
– on the participant of a relationship type				
	–	–	√	–
· irregular and heterogeneous structure				
– variant-valued attribute constructor				
	√	–	√	√
– disjunctive constraints on relationship types				
	–	√	√	√
· document-centric data				
– basic mixed content				
	–	–	–	–
– generalized mixed content				
	–	–	–	–
· reuse of content				
– IS-A or the category concept				
	√	–	√	√
– named types and groups of concepts				
	–	–	–	–
· integration of conceptual schemata				
– modeling constructs				
	–	–	√	–
– algorithms for merging schemata				
	–	–	√	–
– algorithms for the data translation between schemata				
	–	–	√	–

[1] indirect modeling using hierarchical relationship types is possible

Acknowledgement

This research was supported by the National programme of research (Information society project 1ET100300419).

References

[1] A. Badia. Conceptual Modeling for Semistructured Data. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering Workshops (WISE 2002 Workshops), p. 170-177.* Singapore, December 2002.

[2] Y.B. Chen, T.W. Ling, M.L. Lee, A Case Tool for Designing XML Views. In *Proceedings of the Second International Workshop on Data Integration over the Web (DIWeb'02), p. 47-57.* Toronto, Canada, May 2002.

[3] Y.B. Chen, T.W. Ling, M.L. Lee. Automatic Generation of XQuery View Definitions from ORA-SS Views. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER 2003), p. 158-171.* Chicago, Illinois, USA, October 2003.

[4] Y.B. Chen, T.W. Ling, M.L. Lee. Designing Valid XML Views. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER 2002), p. 463-477*, Tampere, Finland, October 2002.

[5] J. Clark. *XSL Transformations (XSLT) Version 1.0.* World Wide Web Consortium, Recommendation REC-xslt-19991116. November 1999.

[6] J. Clark, M. Makoto. *RELAX NG Specification.* Oasis. December 2001.

[7] G. Dobbie, W. Xiaoying, T.W. Ling, M.L. Lee. ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. *Technical Report, Department of Computer Science, National University of Singapore.* December 2000

[8] G. Dobbie, W. Xiaoying, T.W. Ling, M.L. Lee. Designing Semistructured Databases Using ORA-SS Model. In *Proceedings of the 2nd International Conference on Web Information Systems Engineering (WISE'01), p. 171-182.* Kyoto, Japan, December 2001.

[9] D.W. Embley, S.W. Liddle, R. Al-Kamha. Enterprise Modeling with Conceptual XML. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER 2004), p. 150-165.* Shanghai, China, November 2004.

[10] D.W. Embley, W.Y. Mok. Developing XML documents with guaranteed 'good' properties. In *Proceedings of the 20th International Conference on Conceptual modeling (ER 2001), p. 426-441.* Yokohama, Japan, November 2001.

[11] L. Feng, E. Chang, T. Dillon. A Semantic Network-Based Design Methodology for XML Documents. *ACM Transactions on Information Systems, Volume 20, Number 4, p. 390-421.* October 2002.

[12] D. C. Fallside, P. Walmsley. *XML Schema Part 0: Primer Second Edition.* World Wide Web Consortium, Recommendation REC-xmlschema-0-20041028. October 2004.

[13] T. Halpin. *Information Modeling and Relational Databases From Conceptual Analysis to Logical Design.* Morgan Kaufmann Publishers, 2001. ISBN: 1-55860-672-6

[14] International Organization for Standardization. *Information Technology Document Schema Definition Languages (DSDL) Part 3: Rule-based Validation Schematron.* ISO/IEC 19757-3, February 2005.

[15] B.R. Loscio, A.C. Salgado, L.R. Galvao. Conceptual Modeling of XML Schemas. In *Proceedings of the Fifth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2003), p. 102-105.* New Orleans, Louisiana, USA, November 2003.

[16] M. Mani. EReX: A Conceptual Model for XML. In *Proceedings of the Second International XML Database Symposium (XSym 2004), p. 128-142.* Toronto, Canada, August 2004.

[17] M. Mani. Data Modeling Using XML Schemas. PhD Dissertation, *http://web.cs.wpi.edu/ mmani/dissertation.pdf*, July 2003.

[18] M. Mani, D. Lee, R.R.Muntz. Semantic Data Modeling Using XML Schemas. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001), p. 149-163.* Yokohama, Japan, November 2001.

[19] F. Manola, E. Miller. *RDF Primer.* World Wide Web Consortium, Recommendation REC-rdf-primer-20040210. February 2004.

[20] P. Marinelli, C. S. Coen, F. Vitali. SchemaPath, a Minimal Extension to XML Schema for Conditional Constraints. In *Proceedings of the 13th International Conference on World Wide Web (WWW 2004), p. 164-174.* New York, USA, May 2004.

[21] Object Management Group. *UML 2.0 Superstructure Specification.* October 2004.

[22] G. Psaila. ERX: A Conceptual Model for XML Documents. In *Proceedings of the 2000 ACM Symposium on Applied Computing, p. 898-903.* Como, Italy, March 2000.

[23] A. Sengupta, S. Mohan, R. Doshi. XER - Extensible Entity Relationship Modeling. In *Proceedings of the XML 2003 Conference, p. 140-154.* Philadelphia, USA, December 2003.

[24] M. K. Smith, Ch. Welty, D. L. McGuinness. *OWL Web Ontology Language Guide.* World Wide Web Consortium, Recommendation REC-owl-guide-20040210. February 2004.

[25] B. Thalheim. *Entity-Relationship Modeling: Foundations of Database Technology.* Springer Verlag, 2000, Berlin, Germany. ISBN: 3-540-65470-4

[26] X. Yang, T.W. Ling, M.L. Lee. Resolving Structural Conflicts in the Integration of XML Schemas: A Semantic Approach. In *Proceedings of the 22nd International Conference on Conceptual Modeling (ER 2003), p. 520-533.* Chicago, Illinois, USA, October 2003.