

XSEM - A Conceptual Model for XML

Martin Necasky

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

Abstract

We propose a new conceptual model for XML data called *XSEM* as a combination of several approaches in the area of the conceptual modeling for XML. The model divides the conceptual modeling process of XML data to two levels. On the first level, a designer designs an overall non-hierarchical conceptual schema of a domain. On the second level, he or she derives different hierarchical representations of parts of the overall conceptual schema using transformation operators. These hierarchical representations describe how the data is organized in an XML form.

Keywords: Conceptual modeling, XML, XML Schema

1 Introduction

Recently, XML is used for an exchange of structured and semistructured data between different information systems. Moreover, it is frequently applied as a format for an internal data representation and as a logical format for storing data into databases. As XML gets near to the core of applications modeling of XML data should become an inseparable part of modeling of application data on the conceptual level.

Today, application data are modeled on the conceptual level using for example the E-R model or UML. Often, there is a central overall conceptual schema describing the application domain and several XML schemes describing views on parts of data described by the conceptual schema.

Assume for example a medical application collecting data about patients and physicians from several hospitals and separate clinics through Internet. There is an overall conceptual schema in the E-R model describing the application domain. The application collects the data in XML. There are several XML schemes describing structures of XML documents for collecting data about patients, examinations, diagnoses, treatments, and so on. For example, if a physician from a hospital diagnoses a disease of a patient, an XML message with information about the diagnose must be send to the central application. Hence, a hospital system must create an XML document in

the required form and send it to the central application. The central application receives the diagnose in XML and transforms it into its internal representation.

Today, structures of XML data are designed separately from the overall conceptual schema. Moreover, schemes for the XML data are designed on the logical level using XML Schema specified by Fallside *et al.* (2004) or another XML schema language. These languages can not describe required semantics of the data, design of more complex data is not easy to survey, and there is no interconnection between the structure of XML documents and the overall conceptual schema.

If we want to model XML on the conceptual level we have to deal with some special features of XML data. The main features are the following: hierarchical and irregular structure, ordering, and mixed content. Hence, the existing conceptual models are not suitable for the conceptual modeling of XML data without any further extensions.

There are some approaches, for example ERX proposed by Psaila *et al.* (2000) or XER proposed by Sengupta *et al.* (2003), extending the E-R model to be suitable for the conceptual modeling of XML data. However, E-R schemes are not hierarchical and there is a problem with modeling of a hierarchical structure of XML data.

Pigozzo *et al.* (2005) propose an algorithm for transforming general E-R schemes (containing $M : N$ and n -ary relationship types) to hierarchical XML schemes. Similarly, Bird *et al.* (2000) propose an algorithm for transforming ORM schemes to hierarchical XML schemes. However, the algorithms derive an XML schema automatically without following user's requirements on how the data should be organized in hierarchical XML documents.

Duta *et al.* (2004) propose an algorithm for transforming relational schemes to hierarchical XML schemes. Several possible hierarchical representations of a relation are derived and the best representation is selected. The selection is based on criteria such as the nested and compact structure, and the length of XML data file.

Another possibility of how to model XML data is to start from a hierarchical structure. This approach may be called *hierarchical approach*. There are conceptual models based on the hierarchical approach, for example ORA-SS proposed by Dobbie *et al.* (2000). Using this approach we can model hierarchical structures easily. However, a problem with modeling of non-hierarchical structures arises. Moreover, hierarchical schemes are not so transparent as non-hierarchical E-R schemes. A designer must think about the data in hierarchies which is not natural in every situation.

The problem of the approaches is that it is not possible to design an overall non-hierarchical conceptual

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

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM 2007), Ballarat, Victoria, Australia, January 2007. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 67. John F. Roddick and Annika Hinze, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

schema of the domain and derive several hierarchical organizations of parts of the schema describing required XML documents on the conceptual level as required by the example medical application. Moreover, it is not possible to derive different hierarchical organizations of the same parts of the overall conceptual schema.

Necasky (2006) offers a survey of the conceptual modeling for XML. He proposes a detailed list of requirements for conceptual models for XML, describes several conceptual models for XML in a unified formalism, and compares the described models against the requirements.

In this paper we propose a conceptual model for XML data called *XSEM*. The main idea of our work is to divide the XML conceptual modeling process to two parts. The first part consists of designing an overall conceptual schema using an extension of the classical E-R model proposed by Chen called *XSEM-ER*. The second part consists of designing a hierarchical organization of the structures from the first part using a hierarchical model called *XSEM-H*. The hierarchical organization is derived from the XSEM-ER level by *operators* for the transformation of non-hierarchical XSEM-ER constructions to hierarchical XSEM-H constructions. It is possible to design more than one hierarchical organization of the same data modeled by an XSEM-ER schema.

There are approaches such as WebML proposed by Ceri *et al.* (2000) based on a similar idea where data is modeled on two levels. On the first level, a non-hierarchical schema is designed using the E-R model. On the second level, a structure for presentation of the data on a web site is derived from the first level schema. However, these approaches are designed for modeling a structure of web sites and not for modeling XML data in general.

The rest of the paper is organized as follows. In Section 2, we describe modeling constructs of the XSEM-ER model. In Section 3, we describe constructs of the XSEM-H model. In Section 4, we describe the basic operators for the transformation of parts of XSEM-ER schemes to their hierarchical representation described by XSEM-H schemes.

2 XSEM-ER Model

The XSEM-ER model is the first level of the XSEM model. It is an extension of the classical E-R model proposed by Chen. Each schema in the E-R model is also a schema in the XSEM-ER model. It means, that the operators for transformations to the XSEM-H level are applicable to E-R schemes directly. Hence, a designer familiar with the E-R model, but the one who is not familiar with the extending XSEM-ER concepts or the one who does not need these concepts, can use only the constructs of the E-R model.

2.1 Basic Modeling Constructs

Basic modeling constructs of the XSEM-ER model are the same as in the case of the E-R model. There are strong and weak entity types and relationship types connecting two or more entity types.

Attributes. Attributes represent data properties of real-world objects and relationships. There are *simple*, *composite*, and *multivalued* attributes. A *simple* attribute A has assigned a domain $dom(A)$ of simple values. If A_1, \dots, A_n are attributes then $A(A_1, \dots, A_n)$ is a *composite attribute* with a domain $dom(A) = dom(A_1) \times \dots \times dom(A_n)$. If A is an attribute then $A[m, n]$ is a *multivalued* attribute with a domain $dom(A[m, n]) = \{ \langle a_1, \dots, a_k \rangle : m \leq k \leq n, \forall 1 \leq i \leq k, a_i \in dom(A) \}$ containing ordered lists

of values from $dom(A)$. We denote an empty list by \perp . Instead of $A[0, 1]$, we can use $A?$ called *optional* attribute.

Entity Types. Entity types are basic building blocks of XSEM-ER schemes. They represent real-world objects like persons, buildings, or cars. XSEM-ER entity types are similar to E-R entity types. However, there are some differences between them. These differences result from natural ordering of XML data.

The first difference is that a list of attributes of an XSEM-ER entity type is ordered. Further, we can use ordering as an implicit identification and we do not need to specify explicit keys. Thus, an XSEM-ER entity type can have an empty key or it can have a non-empty key composed of optional attributes. In the both cases, there can be two different instances of the same entity type without a key value and having the same values of attributes. For their identification, implicit ordering is used.

Formally, an *entity type* is a pair $E = (attr(E), id(E))$ where E is a name of the entity type, $attr(E)$ is a list of attributes of the entity type, and $id(E)$ is a set of some attributes from $attr(E)$ called *key* of the entity type. The key can be empty and it may be composed only of simple or optional attributes, or composite attributes composed (recursively) only from simple and optional attributes.

An entity type E represents a set of individuals described by the entity type. This set is called *entity set* of E and it is denoted as E^C . Each individual in E^C is called *entity* of the entity type E . Each entity e of E has assigned values of the attributes from $attr(E)$ by a tuple function defined on $attr(E)$. $\forall A \in attr(E)$ the value of A assigned to e by the tuple function is denoted as $e(A)$ where $e(A) \in dom(A)$. If A is multivalued and the tuple function assigns an empty sequence \perp to e as a value of A then we say that e has not a value of A .

The following condition must be satisfied for each two entities $e_1, e_2 \in E^C$.

$$\begin{aligned} & [(\exists A \in id(E)) (e_1(A) \neq \perp \wedge e_2(A) \neq \perp) \\ & \wedge (\forall B \in id(E)) (e_1(B) = e_2(B))] \\ & \rightarrow (e_1 = e_2) \end{aligned}$$

The condition is called *weak key condition*. It says that if there is an attribute $A \in id(E)$ such that the both e_1 and e_2 have a value of A and e_1 and e_2 equals on values of the all attributes from $id(E)$ then they are the same entity.

For example, assume an entity type *Section* modeling sections of papers. *Section* has a key composed of an attribute *label*. However, there can be sections without a value of *label*. Hence, the attribute must be optional. The key condition is applied only to the sections having a value of *label*. There can not be two different sections having the same value of *label*. However, there can be two different sections without a value of *label* and with the same values of the rest of the attributes. The sections with a value of *label* can be identified in the paper by this value. The sections without a value of *label* can be identified by implicit ordering, because they are ordered. Example 2.1 shows a paper with sections in XML. There are three sections. Each of them can be identified by implicit ordering (the first, second, and third section in the paper). Moreover, the second section can be identified explicitly by its label.

In a graphical representation, an entity type is displayed as a box with a name of the entity type and a list of its attributes. There is a filled circle displayed before the name of each of the key attributes. If there are two or more attributes composing the key, their filled circles are connected by a solid line.

```

<paper><title>XSEM - ...</title>
  <section><title>Introduction</title>
  ...
</section>
<section><title>XSEM-ER Model</title>
  <label>xsem_er_model</label>
  ...
</section>
<section><title>XSEM-H Model</title>
  ...
</section>
</paper>

```

Example 2.1: Paper with sections in XML

Figure 2.1 shows examples of entity types. There are entity types *Hospital*, *Physician*, and *Patient* modeling hospitals, physicians, and patients. Each of them has a key composed of an attribute *name*.

Relationship Types. Relationship types represent relationships between real-world objects. For example, physicians are employed at departments of hospitals, people own cars, etc. As in the case of entity types, relationship types are similar to E-R relationship types, but there are some differences resulting from natural ordering of XML data.

Again a list of attributes of an XSEM-ER relationship type is ordered and we can use ordering to distinguish instances of a relationship type. In E-R, we distinguish instances of a relationship type by values of the entity types E_1, \dots, E_n composing the relationship type. Hence, given instances e_1, \dots, e_n of the entity types can compose only one instance of the relationship type in E-R. However, e_1, \dots, e_n can compose more than one instance of the relationship type in XSEM-ER and we can use ordering of XML data to distinguish the instances.

Formally, a *relationship type* is a pair $R = (attr(R), part(R))$ where R is a name of the relationship type, $attr(R)$ is a list of attributes of the relationship type, and $part(R)$ is a set of two or more pairs (E_i, l_i) where E_i is an entity type called *participant* of R and l_i is a label called *role* of E_i in R which can be empty. There can not be $i \neq j$ such that $(E_i, l_i) = (E_j, l_j)$.

Similarly to entity types, a relationship type R represents a set of *relationships* of R called *relationship set* of R denoted as R^C . Each relationship r of R has assigned values of the attributes from $attr(R)$ and participants from $part(R)$ by a tuple function defined on $attr(R) \cup part(R)$. $\forall A \in attr(R)$ the value of A assigned to r by the tuple function is denoted as $r(A)$ where $r(A) \in dom(A)$. Similarly, $\forall (E, l) \in part(R)$ the value of (E, l) assigned to r is denoted as $r((E, l))$ where $r((E, l)) \in E^C$.

In a graphical representation a relationship type is displayed as a hexagon with a name of the relationship type and a box under the hexagon with a list of its attributes. Participants of the relationship type are connected to the hexagon by solid arrows.

Figure 2.1 shows a binary relationship type *Employ* with participants *Physician* and *Department*. We model by this relationship type that physicians are employed at departments of hospitals.

Weak Entity Types. Often, an entity type depends on one or more another entity types. For example, we can model departments of hospitals by an entity type *Department*. However, when speaking about a department we have to specify a hospital of the department. In such cases we can use weak entity types.

Formally, a *weak entity type* is a triple $E =$

$(attr(E), id(E), det(E))$ where $id(E)$ is called *relative key* of E and $det(E)$ is a set of one or more pairs (E_i, l_i) where E_i is an entity type called *determinant* of E and l_i is a label called *role* of E_i in E which can be empty. There can not be $i \neq j$ such that $(E_i, l_i) = (E_j, l_j)$.

Each entity e of E has assigned values of the attributes from $attr(E)$ and determinants from $det(E)$ by a tuple function defined on $attr(E) \cup det(E)$. $\forall A \in attr(E)$ the value of A assigned to e is denoted as $e(A)$ where $e(A) \in dom(A)$ and $\forall (E', l) \in det(E)$ the value of (E', l) assigned to e is denoted as $e((E', l))$ where $e((E', l)) \in E^C$.

The weak key condition defined for entity types must be satisfied for a weak entity type too, but only in the context of entities having the same values of determinants.

An entity type with an empty set of determinants is equivalent to the former entity types without determinants. Such entity types are called *strong entity types*.

There should not be a sequence of entity types E_1, \dots, E_n such that $E_1 = E_n$ and $\forall 1 \leq i < n$ E_{i+1} is a determinant of E_i . Hence, we use *order* of entity types proposed by Thalheim (2000). A strong entity type has an order 0. A weak entity type E has an order $k+1$, where k is an order of a determinant of E with the highest order. Each entity type must have an order. Hence, the previous sequence E_1, \dots, E_n can not appear in a schema.

Weak entity types proposed in this section are similar to higher order relationship types proposed by Thalheim (2000). However, we do not unify the notions of weak entity types and relationship types according to Thalheim. The reason is that users are used to distinguish these notions and the semantics of the notions. On the other hand, such a unification would simplify the formalism used in this paper. Hence, we are going to study the unification in the further research.

In a graphical representation, a weak entity type is displayed in the same way as an entity type, only a hexagon is added into the box. Determinants are connected to the box by solid arrows.

At Figure 2.1, each hospital has departments modeled by a weak entity type *Department* with a determinant *Hospital*. Moreover, patients visit physicians at departments of hospitals. Such visits are modeled by a weak entity type *Visit* with determinants *Patient*, *Physician*, and *Department*.

Visit has a key composed of a required attribute *date*. Hence, each *Visit* entity must have a value of *date* which must be unique in the context of *Visit* entities having the same values of determinants.

It is possible to model visits by a relationship type. However, we need *Visit* to have a relative key *date*. Moreover, *Visit* is used as a determinant of another weak entity types in the following examples.

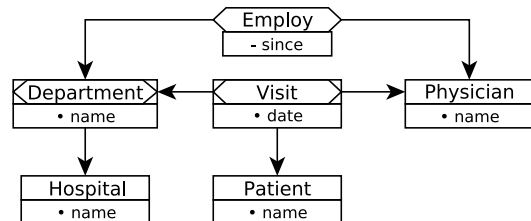


Figure 2.1: XSEM-ER Entity and Relationship Types

2.2 Extending Modeling Constructs

In this section, we describe new modeling constructs extending the constructs of the E-R model. We propose *data node types* and *cluster types* for modeling semistructured, irregular, and heterogeneous data.

Data Node Types. For modeling semistructured data we propose *data node types*. A data node type represents unstructured data values assigned to a given entity. These values are not attribute values of the entity. They are data values which are mixed with the relationships having the entity as a participant value and the weak entities having the entity as a determinant value.

Formally, a *data node type* is a pair $D = (type(D), par(D))$ where D is a *name* of the data node type, $type(D)$ is a data type restricting possible values of the data node type, and $par(D)$ is an entity type called *parent* of the data node type.

In a graphical representation, a data node type is displayed as an ellipse with a name of the data node type. We display a data type of the data node type only when necessary. In such a case, the data type is displayed at the name of the node.

For example, assume a patient visiting a physician. During the visit, the physician writes a description of the course of the visit. The description is not an attribute value of the visit, it is an unstructured text assigned to the visit. Moreover, parts of the description are mixed with examinations and anamneses made during the visit. To model such descriptions we use the data node type *VisitDesc* at Figure 2.2. Example 2.3 shows a possible XML representation of a visit with an unstructured text description mixed with an examination and anamnesis made during the visit.

However, it is not enough to propose only the notion of data node types. We need a possibility to specify that only a part of relationships and weak entities of a *Visit* instance is mixed with an unstructured data. Moreover, ordering between parts of the unstructured data, relationships, and weak entities is important. Hence, we need further modeling constructs. These constructs are proposed in the following text.

Outgoing Cluster Types. We propose a notion of *outgoing cluster types* for modeling irregular and heterogeneous data. We evolve it from the notion of HERM cluster types proposed by Thalheim (2000).

An outgoing cluster type represents a union of entity types. It can be used as a participant of another outgoing cluster type, as a participant of a relationship type, or as a determinant of a weak entity type.

Formally, let P_1, \dots, P_n be a list of entity types and outgoing cluster types. An *outgoing cluster type* C with the participants P_1, \dots, P_n is an expression $P_1 + \dots + P_n$. A cluster set C^C of the cluster is defined as $\bigcup_{i=1}^n P_i^C$.

In a graphical representation, an outgoing cluster type is displayed as a circle with an inner label $+$. It is connected by a solid line with a relationship type or weak entity type it participates in. Each participant of the cluster type is connected by an arrow going from the circle to the participant.

Using outgoing cluster types we can model irregular structure of XML. Assume for example the weak entity type *Visit* at Figure 2.1. It means that patients visit physicians at departments of hospitals. However, patients can visit physicians at separate clinics too. This is an example of irregular structure we can express in XML. We can use an outgoing cluster type *Department + Clinic* to model this situation. The cluster type is shown at Figure 2.2. Example 2.2 shows a possible XML representation of visits of a patient with a name "John Black". There are two patient's visits in the XML document. First, he visited

a physician "Bill White" at a department "Department A" of a hospital "Hospital B". Then he visited a physician "Jack Brown" at a clinic "Clinic C".

```
<patient>
  <name>John Black</name>
  <visit><date>2006-09-12</date>
    <physician>
      <name>Bill White</name>
    </physician>
    <department><name>Department A</name>
    <hospital>
      <name>Hospital B</name>
    </hospital>
  </department>
</visit>
  <visit><date>2006-10-03</date>
    <physician>
      <name>Jack Brown</name>
    </physician>
    <clinic>
      <name>Clinic C</name>
    </clinic>
  </visit>
</patient>
```

Example 2.2: Irregular structure in XML

With the notion of outgoing cluster types, an entity type can participate in a relationship type indirectly, by means of an outgoing cluster type. Hence, we define the notion of *recursive participants*.

Assume an outgoing cluster type C with a participant P' . We say that an entity type or outgoing cluster type P is a *recursive participant of C by means of P'* if $P = P'$ or P' is an outgoing cluster type having P as a recursive participant. Now assume a relationship type R with a participant P'' . We say that an entity type or outgoing cluster type P is a *recursive participant of R by means of P''* if $P = P''$ or P'' is an outgoing cluster type having P as a recursive participant.

A *recursive determinant* of a weak entity type is defined in the same way as in the case of recursive participants of relationship types. The definition of order of a weak entity type must be modified. A weak entity type E has an order $k+1$, where k is an order of a recursive determinant of E with the highest order.

Incoming Cluster Types. We use *incoming cluster types* for grouping different relationship types, weak entity types, and data node types having the same participant/determinant/parent called *parent* of the incoming cluster type. Groups of relationship types, weak entity types, and data node types are used for modeling mixed content. Moreover, ordering can be specified on such groups.

Formally, let E be an entity type and P_1, \dots, P_n be a list of relationship types, weak entity types, and data node types each having E as a participant, determinant, or parent. An *incoming cluster type* is an expression $(E, P_1 + \dots + P_n)$.

In a graphical representation, an incoming cluster type is displayed as a circle with an inner label $+$. It is connected by a solid line with its parent. P_1, \dots, P_n are connected by an arrow going to the circle.

Together with data node types and ordering constraints (proposed in the next section) we use incoming cluster types for modeling mixed content in XML documents. For example, we can use the incoming cluster type $(Visit, Examination + Anamnesis + VisitDesc)$ at Figure 2.2 to model a description of a visit mixed with the examinations and anamnesis made during the visit. Example 2.3 shows an XML representation of such a description. The XML docu-

The following unary hierarchical projection completes the decomposition of *Visit*:

$Visit^{PatientPhysician}[Visit \rightarrow Clinic + Department]$

These hierarchical projections describe the structure of an XML document at Example 2.4. There is the patient "John Black" and the physicians "Bill White" and "Jack Brown" he visited. For each physician there is a list of John Black's visits of the physician.

```
<patient>
  <name>John Black</name>
  <physician><name>Bill White</name>
    <visit><date>2006-09-12</date>
      <department><name>Department A</name>
        <hospital>
          <name>Hospital B</name>
        </hospital>
      </department>
    </visit>
    <visit><date>2006-10-19</date>
      <clinic>
        <name>Clinic C</name>
      </clinic>
    </visit>
  </physician>
  <physician><name>Jack Brown</name>
    <visit><date>2006-10-03</date>
      <clinic>
        <name>Clinic C</name>
      </clinic>
    </visit>
  </physician>
</patient>
```

Example 2.4: XML Described by Hierarchical Projections

The following hierarchical projections describe the structure of the XML document at Example 2.2:

$Visit[Patient \rightarrow Visit]$,
 $Visit^{Patient}[Visit \rightarrow Physician]$,
 $Visit^{Patient}[Visit \rightarrow Clinic + Department]$

Data node types are similar to weak entity types with one determinant. Hence, a unary hierarchical projection of a data node type can be defined in a similar way. It is possible to specify only the following hierarchical projection of a data node type D .

$$D[par(D) \rightarrow D]$$

2.4 Integrity Constraints

Specification of integrity constraints is an inseparable part of the conceptual modeling process. In this section, we extend the notion of *cardinality constraints* known from the E-R model and propose a notion of *ordering constraints*.

Cardinality Constraints. Cardinality constraints known from the E-R model are too weak to use for modeling XML. It is because we need to transform a non-hierarchical structure of XSEM-ER schemes to a hierarchical structure. Hence, we need more information about cardinalities of relationship types, or weak entity types, respectively, having more than 2 participants, or determinants, respectively.

Thalheim (2000) proposes a more powerful kind of cardinality constraints called *projected cardinality constraints*. We propose a similar type of cardinality

constraints called *cardinality constraints with a context* that we use to constraint cardinalities of hierarchical projections of relationship types and weak entity types.

Formally, let ρ be a hierarchical projection of R to $P \rightarrow Q$ with a context P'_1, \dots, P'_k . A *cardinality constraint with a context* has the form

$$card(\rho, P) = (m, n), \text{ resp. } card(\rho, Q) = (m, n)$$

If $k = 0$ (i.e. an empty context) then it specifies that for an instance p of P , or q of Q , respectively, the number of triples from ρ^C with a parent p , or child q , respectively, is in the range (m, n) , including m, n . If $k > 0$ (i.e. a non-empty context) then it specifies that for instances p'_1, \dots, p'_k of P'_1, \dots, P'_k and for an instance p of P , or q of Q , respectively, such that p , or q , respectively, appears in R with a context p'_1, \dots, p'_k , the number of triples from ρ^C with a context p'_1, \dots, p'_k and with a parent p , or child q , respectively, is in the range (m, n) , including m, n . If $P = R$ or $Q = R$ then the corresponding constraint must be $(1, 1)$.

As an example, we specify the following cardinality constraints for the schema at Figure 2.2. The cardinality constraint

$$\rho_1 = Department[Hospital \rightarrow Department]$$

$$card(\rho_1, Hospital) = (1, *)$$

specifies that there are one or more departments in a hospital. The cardinality constraint

$$\rho_2 = Anamnesis[Visit \rightarrow Anamnesis]$$

$$card(\rho_2, Visit) = (0, 1)$$

specifies that there is none or one anamnesis made during a visit. Finally, the cardinality constraint

$$\rho_3 = Visit^{Clinic+Department}[Physician \rightarrow Patient]$$

$$card(\rho_3, Physician) = (1, 100)$$

specifies that if patients visit a physician at a given department or clinic then the number of patients visiting the physician at the department or clinic is between 1 and 100.

Ordering. An important extending concept in XSEM-ER is the concept of ordering. We describe two types of ordering. The first type specifies ordering on hierarchical projections of relationship types and the second type specifies ordering on incoming cluster types.

Ordering on Hierarchical Projections of Relationship Types. Let R be a relationship type and ρ be a hierarchical projection of R with a context P'_1, \dots, P'_k and with a parent P . *Ordering on the hierarchical projection* ρ has the form *ordered*(ρ) and specifies that for instances p'_1, \dots, p'_k of P'_1, \dots, P'_k and for an instance p of P the set of triples from ρ^C with a context p'_1, \dots, p'_k and with a parent p is linearly ordered.

Ordering on Incoming Cluster Types. Assume an entity type E and its incoming cluster type C . If C is ordered, denoted as *ordered*(C), then for an instance e of E the set of all instances of relationship types, weak entity types, and data node types in C having e as a corresponding participant, determinant, or parent value is linearly ordered.

When there is an ordering specified on an incoming cluster type, the cluster type is displayed with a thick line.

As an example, we specify the following ordering constraints for the schema at Figure 2.2. The constraint

$$\text{ordered}(\text{Visit}[\text{Patient} \rightarrow \text{Physician}])$$

specifies that the list of physicians visited by a given patient is ordered. The constraint

$$\text{ordered}(\text{Visit}, \\ \text{Anamnesis} + \text{Examination} + \text{VisitDesc})$$

specifies that for a given visit the list containing the anamnesis and examinations made during the visit, and parts of the visit description is ordered.

3 XSEM-H Model

The XSEM-H model is the second level of the XSEM model. It allows designers to design required hierarchical organizations of components designed at the first level. A designer can design a hierarchical organization of a whole XSEM-ER schema or only of a part of this schema. It is also possible to design more than one hierarchical organization of the same part of the XSEM-ER schema.

An XSEM-H schema is only a specification of a hierarchical organization of a part of a given XSEM-ER schema. It does not add any semantics. All the semantics is described on the XSEM-ER level. Hence, we can comprehend an XSEM-H schema as a *hierarchical view* of the corresponding part of the XSEM-ER schema.

XSEM-H schema. An XSEM-H schema is an *oriented graph* where nodes are *hierarchical entity types* and *hierarchical data node types*, and edges are *hierarchical relationship types*. Because hierarchical relationship types are graph edges, they are only binary. Hierarchical entity types represent entity types and relationship types from the XSEM-ER level, and hierarchical relationship types represent hierarchical projections of relationship types and weak entity types from the XSEM-ER level. Moreover, XSEM-H contains modeling constructs called *clusters of hierarchical entity types* and *clusters of hierarchical relationship types* for a hierarchical representation of cluster types from XSEM-ER.

Formally, an *XSEM-H schema* is a pair

$$\mathcal{H} = (\text{compon}(\mathcal{H}), \text{bound}(\mathcal{H}))$$

where $\text{compon}(\mathcal{H})$ is a set of hierarchical entity types, relationship types, and clusters, and $\text{bound}(\mathcal{H})$ is an XSEM-ER schema. We say that \mathcal{H} is *bounded* with $\text{bound}(\mathcal{H})$.

Figure 3.1 shows an example of an XSEM-H schema. It is a hierarchical view of the XSEM-ER schema at Figure 2.2.

Hierarchical Entity Types. A *hierarchical entity type* is a pair

$$E^H = (\text{hier}(E^H), \text{bound}(E^H))$$

where E^H is a *name* of the hierarchical entity type, $\text{hier}(E^H)$ is an ordered list of hierarchical relationship types and clusters of hierarchical relationship types having E^H as a parent, and $\text{bound}(E^H)$ is an entity type, relationship type, or a hierarchical projection of a relationship type. We say that E^H is bounded with $\text{bound}(E^H)$.

Figure 3.1 shows examples of hierarchical entity types. Each hierarchical entity type E^H is displayed as a box containing $\text{bound}(E^H)$. For example, there is a hierarchical entity type bounded

with the entity type *Physician* or a hierarchical entity type bounded with a hierarchical projection $\text{Employ}^{\text{Physician}}[\text{Clinic} \rightarrow \text{Employ}]$ (we do not show the context at the box at the figure).

Hierarchical Data Node Types. To represent data node types the model contains hierarchical data node types. It is defined in the similar way as hierarchical entity types.

A *hierarchical data node type* is a construction

$$D^H = (\text{bound}(D^H))$$

where D^H is a *name* of the hierarchical data node type and $\text{bound}(D^H)$ is a data node type, D^H is bounded with.

At Figure 3.1 there is a hierarchical data node type bounded with the data node type *VisitDesc*.

Hierarchical Relationship Types. Hierarchical relationship types are oriented edges representing hierarchical organization of components from an XSEM-ER schema.

Formally, a *hierarchical relationship type* is a 4-tuple

$$R^H = (P^H, Ch^H, \rho, l)$$

where R^H is a *name* of the hierarchical relationship type, P^H is a hierarchical entity type or a cluster of hierarchical entity types called *parent participant* of R^H , Ch^H is a hierarchical entity type, a hierarchical data node type, or a cluster of hierarchical entity types called *child participant* of R^H , ρ is a hierarchical projection of a relationship type, weak entity type, or data node type R with a parent P and a child Ch , and l is a string from a set of strings L called *label* of R^H . We say that R^H is *bounded* with R by the meaning of ρ .

The label l can be empty when the child participant is a data node type. This label is used on the logical level to mark the logical representation of each instance of the child participant of R^H (it is equivalent to names of XML elements).

If there is a hierarchical entity type which is not a child of any hierarchical relationship type, it is called *root* of the corresponding schema. For each root of the schema the label for the logical representation must be specified directly.

Integrity constraints specified for ρ on the XSEM-ER level (i.e. cardinality constraints and ordering constraints) are directly applied to the hierarchical relationship type R^H . We do not need to specify any special integrity constraints on the XSEM-H level.

Figure 3.1 shows examples of hierarchical relationship types. For example, a hierarchical relationship type $R1$ is bounded with a hierarchical projection

$$\text{Department}[\text{Hospital} \rightarrow \text{Department}]$$

It forms a hierarchical representation of the weak entity type *Department* from the corresponding XSEM-ER schema. Hierarchical relationship types $R2$, $R3$, and $R4$, respectively, form a hierarchical representation of the weak entity type *Visit*. They are bounded with

$$\text{Visit}[\text{Clinic} + \text{Department} \rightarrow \text{Physician}],$$

$$\text{Visit}^{\text{Clinic}+\text{Department}}[\text{Physician} \rightarrow \text{Patient}],$$

$$\text{Visit}^{\text{Clinic}+\text{Department},\text{Physician}}[\text{Patient} \rightarrow \text{Visit}]$$

Clusters of Hierarchical Entity and Relationship Types. Not only entity types, data node types, and relationship types compose XSEM-ER schemes. There are also outgoing and incoming cluster types. That is why we need to propose similar constructs on

the XSEM-H level. There are *clusters of hierarchical entity types* and *clusters of hierarchical relationship types* in XSEM-H. The former represent XSEM-ER outgoing cluster types and the other represent XSEM-ER incoming cluster types.

Formally, a *cluster of hierarchical entity types* has the form $(P_1^H, l_1) + \dots + (P_n^H, l_n)$, where $\forall 1 \leq i \leq n$ P_i^H is a hierarchical entity type or another cluster of hierarchical entity types and l_i is a string from a set of strings L called *label* of P_i^H in the cluster which can be empty.

A *cluster of hierarchical relationship types* has the form $P_1^H + P_2^H + \dots + P_n^H$, where $\forall 1 \leq i \leq n$ P_i^H is a hierarchical relationship type or another cluster of hierarchical relationship types.

At Figure 3.1, a hierarchical relationship type $R8$ has a cluster of hierarchical entity types as a child. There is also a cluster of hierarchical relationship types grouping $R5$, $R6$, and $R7$. These clusters are results of the transformation of the cluster types in the XSEM-ER schema at Figure 2.2.

References. On the instance level of a given XSEM-H schema there can be many redundancies resulting from many-to-many cardinalities of hierarchical projections. For example, a physician can be visited by patients at more than one department or clinic and patients can visit more than one physician at a department or clinic.

In the hierarchical representation modeled by the XSEM-H schema at Figure 3.1, information about a physician is repeated for each department or clinic where the physician was visited by patients. Such information contain values of attributes of the physician and it could contain hierarchical representations of other relationships the physician participates in (if such representations would be included in the schema).

References enable to avoid such redundancies. A reference is a non-hierarchical binary edge in an XSEM-H schema connecting a source hierarchical entity type with a target hierarchical entity type.

Formally, a *reference* has the form $(S^H, T^H)_{ref}$, where S^H, T^H are hierarchical entity types called *source* and *target* of the reference. One of the following conditions must be satisfied: $bound(S^H) = bound(T^H)$, or S^H is bounded with a hierarchical projection ρ , where $parent(\rho) = bound(T^H)$. We say that the source *references* the target.

At Figure 3.1, the child of $R2$ references the parent of $R8$ (displayed as a dashed arrow). It means that on the instance level, in the place of the child of $R2$ it is not necessary to have a representation of the whole instance but only a reference to the corresponding instance of the parent of $R8$. Hence, if a physician is visited by patients at a given department or clinic, there is only a reference to the whole representation of the physician. This whole representation is not repeated. Hence, we avoid the redundancy.

Of course, we do not have to use references to avoid all the redundancies. If the redundancy does not cause problems we can ignore it. Sometimes, for example in a streamed processing of XML, redundancies are necessary.

4 Transformation from XSEM-ER to XSEM-H

After designing an overall conceptual schema on the XSEM-ER level, hierarchical views on the XSEM-H level are designed. A designer derives XSEM-H schemes from the XSEM-ER schema using *transformation operators*. First, the designer selects relationship types and weak entity types from the XSEM-ER

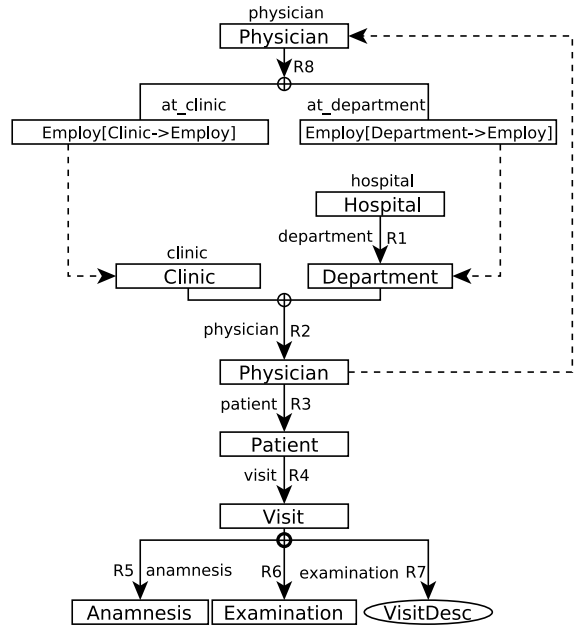


Figure 3.1: XSEM-H Schemes

schema. For each of the selected components its hierarchical representation is constructed using the *operator for a transformation of relationship types and weak entity types*. After this, hierarchical representations are joined together using the *join operator*. In certain cases, a hierarchical relationship type can be contracted and its parent and child can be merged to one hierarchical entity type. This can be done by the *contraction operator*.

We call XSEM-H schemes constructed by the transformation operators as *well-formed XSEM-H schemes*. Only well-formed XSEM-H schemes are allowed in XSEM.

4.1 Operator for a Transformation of Relationship Types and Weak Entity Types

The operator transforms a relationship type or a weak entity type to a hierarchical representation. It takes a relationship type or a weak entity type as an input (with parameters specifying the hierarchical representation) and creates a new XSEM-H schema expressing a hierarchical organization of the component.

XSEM-H schemes constructed by the operator are called *complete full representations* of the corresponding component. There is another type of hierarchical representations called *contracted full representations*. Contracted full representations are created from complete full representations using the contraction operator.

We propose a formal description of the operator for transformations of weak entity types. However, the description of the operator for transformations of relationship types is similar. A simple case is the transformation of data node types.

It is possible to design more than one hierarchical representation of a weak entity type. Hence, the operator has parameters for the specification of the required complete full representation. When using the operator, we must specify the following parameters:

- weak entity type E with determinants denoted as $\{P_1, \dots, P_n\}$ from an XSEM-ER schema \mathcal{ER} and a label l_E
- $k, 0 \leq k \leq n$ and $\forall 1 \leq i \leq n$

- if P_i is a strong entity type then a label l_i must be given
- else if P_i is a weak entity type then a hierarchical representation \mathcal{H}^{P_i} of P_i and a label l_i must be given
- else P_i is an outgoing cluster type, let $Q_{i,1}, \dots, Q_{i,j_i}$ be all the recursive participants of P_i being entity types and $\forall 1 \leq j \leq j_i$
 - * if $Q_{i,j}$ is a strong entity type then a label $l_{i,j}$ must be given
 - * else $Q_{i,j}$ is a weak entity type, hence a hierarchical representation $\mathcal{H}^{Q_{i,j}}$ of $Q_{i,j}$ and a label $l_{i,j}$ must be given

The parameter k specifies that the projection of the weak entity type E to the determinants P_1, \dots, P_k will be represented by hierarchical relationship types bounded with the following binary hierarchical projections

$$E^{P_1 \dots P_{i-1}}[P_i \rightarrow P_{i+1}], 1 \leq i < k$$

The rest of the weak entity type E will be represented by hierarchical relationship types bounded with the following unary hierarchical projections

$$E^{P_1 \dots P_{k-1}}[P_k \rightarrow E], E^{P_1 \dots P_k}[E \rightarrow P_i], k < i \leq n$$

In each complete full representation there is exactly one hierarchical entity type bounded with the weak entity type E . We call this hierarchical entity type as *bottom* of the hierarchical representation.

As an example of applying the operator assume the weak entity type *Visit* from the XSEM-ER schema at Figure 2.2. One of the possibilities of a hierarchical representation of *Visit* is in the XSEM-H schema at Figure 3.1. The schema is constructed by the operator with $k = 3$ (see the following operator showing how to add hierarchical representations of *Anamnesis*, *Examination*, and *VisitDesc*) and:

- $E = \textit{Visit}$ (with a label *visit*),
- $P_1 = \textit{Clinic} + \textit{Department}$ (with labels *clinic* and *department*, a hierarchical representation of *Department* must be supplied), $P_2 = \textit{Physician}$ (with a label *physician*), and $P_3 = \textit{Patient}$ (with a label *patient*)

```

<hospital>
  <name>Hospital B</name>
  <department>
    <name>Department A</name>
    <physician><name>Bill White</name>
    <patient><name>John Black</name>
    <visit><date>2006-09-12</date>
    Because of the family anamnesis
    (<anamnesis>...</anamnesis>)
    there is a suspicion of liver
    cancer. Consequently, I made
    a laboratory examination
    (<examination>...</examination>)
    ...
  </visit>
</patient>
</physician>
</department>
</hospital>

```

Example 4.1: XML Described by an XSEM-H schema

For example, the resulting schema describes XML documents at Example 4.1 and Example 4.2. At Example 4.1 there is the department "Department A" at the hospital "Hospital B" and the physician "Bill White" visited by the patient "John Black" at the department. There is only one such a visit.

At Example 4.2 there is the clinic "Clinic C" and the physicians "Bill White" and "Jack Brown". "Bill White" was visited by the patients "John Black" and "Joe Green" at the clinic (for each of the patient there is only one such a visit). "Jack Brown" was visited by the patient "John Black" at the clinic (there are two such visits).

```

<clinic><name>Clinic C</name>
  <physician><name>Bill White</name>
    <patient><name>John Black</name>
      <visit><date>2006-10-19</date>
      ...
    </visit>
  </patient>
  <patient><name>Joe Green</name>
    <visit><date>2006-09-17</date>
    ...
  </visit>
</physician>
<physician><name>Jack Brown</name>
  <patient><name>John Black</name>
    <visit><date>2006-10-03</date>
    ...
  </visit>
  <visit><date>2006-10-12</date>
  ...
</visit>
</patient>
</physician>
</clinic>

```

Example 4.2: XML Described by an XSEM-H schema

This is not the only hierarchical representation of *Visit*. It is possible to create another representations. Figure 4.1 shows a hierarchical representation of *Visit* constructed by the operator with $k = 1$ and:

- $E = \textit{Visit}$ (with a label *visit*),
- $P_1 = \textit{Patient}$ (with a label *patient*), $P_2 = \textit{Physician}$ (with a label *physician*), and $P_3 = \textit{Clinic} + \textit{Department}$ (with labels *clinic* and *department*)

The operator constructs a hierarchical relationship types $R17$, $R18$, and $R19$, respectively, bounded with hierarchical projections

$$\begin{aligned}
& \textit{Visit}[\textit{Patient} \rightarrow \textit{Visit}], \\
& \textit{Visit}^{\textit{Patient}}[\textit{Visit} \rightarrow \textit{Physician}], \\
& \textit{Visit}^{\textit{Patient}}[\textit{Visit} \rightarrow \textit{Clinic} + \textit{Department}]
\end{aligned}$$

For example, the constructed schema describes the structure of the XML document at Example 2.2.

The algorithm realizing the operator is divided to four phases. The first phase is an initiation. In the second phase, hierarchical entity types representing E and the determinants of E are constructed. In the third phase, hierarchical relationship types bounded with the described hierarchical projections are constructed. When needed, clusters of hierarchical entity types are constructed. Labels supplied as parameters for the operator are used to label constructed hierarchical relationship types and clusters of hierarchical

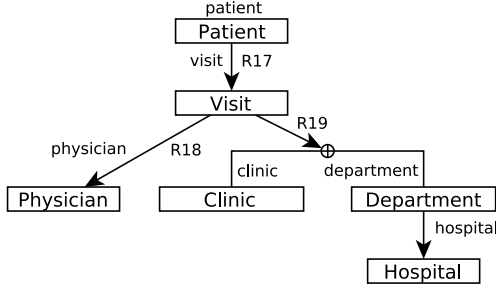


Figure 4.1: XSEM-H Schema

entity types. In the fourth phase, the construction of the hierarchical representation of E is finished.

In the following text, we use a shortened notation $A \square = B$ instead of $A := A \square B$ where \square is a binary operator.

Initiation

- 1: **for** $i := 1$ to n **do**
- 2: **if** P_i is an entity type **then**
- 3: $Q_{i,1} := P_i, l_{i,1} := l_i, j_i := 1$
- 4: **else** $\{P_i$ is an outgoing cluster type $\}$
- 5: $j_i :=$ number of recursive participants of P_i
being entity types
- 6: $Q_{i,1}, \dots, Q_{i,j_i} :=$ a list of recursive participants
of P_i being entity types
- 7: **end if**
- 8: **end for**

Hierarchical entity types construction

- 1: $E^H := ((), E)$
- 2: **for** $i := 1$ to n **do**
- 3: **for** $j := 1$ to j_i **do**
- 4: **if** $Q_{i,j}$ is a strong entity type **then**
- 5: $Q_{i,j}^H := ((), Q_{i,j}), \text{compon}(\mathcal{H}) \cup = \{Q_{i,j}^H\}$
- 6: **else** $\{Q_{i,j}$ is a weak entity type $\}$
- 7: **if** $k = 0$ or $i > 1$ **then**
- 8: $\mathcal{H}^{Q_{i,j}}$ must be a complete full representation
having a hierarchical entity type
bounded with $Q_{i,j}$ as a root
- 9: $Q_{i,j}^H :=$ root from $\mathcal{H}^{Q_{i,j}}$ bounded with $Q_{i,j}$
- 10: **else** $\{k > 0$ and $i = 1\}$
- 11: $B^H :=$ bottom of $\mathcal{H}^{Q_{i,j}}$
- 12: **if** B^H is a hierarchical entity type **then**
- 13: $Q_{i,j}^H := B^H$
- 14: **else** $\{B^H$ is a hierarchical cluster type $\}$
- 15: $Q_{i,j}^H :=$ a copy of B^H (B^H and the copy
share recursively participating hierarchical
entity types, i.e. only the structure of
the cluster is copied, not the hierarchical
entity types)
- 16: **end if**
- 17: **end if**
- 18: $\text{compon}(\mathcal{H}) \cup = \text{compon}(\mathcal{H}^{Q_{i,j}})$
- 19: **end if**
- 20: **end for**
- 21: **end for**

Hierarchical relationship types construction

- 1: **for** $i := 1$ to n **do**
- 2: **if** P_i is an entity type **then**
- 3: $Ch_i^{H,t} := Ch_i^{H,b} := Q_{i,1}^H, \text{label}_i := l_{i,1}$
- 4: **else** $\{P_i$ is an outgoing cluster type $\}$
- 5: **if** $k = 0$ or $i > 1$ **then**

- 6: $Ch_i^{H,t} :=$ a new cluster with participants
 $Q_{i,1}^H, \dots, Q_{i,j_i}^H$ with the same structure as P_i ,
the participants are labeled by $l_{i,1}, \dots, l_{i,j_i}$
- 7: $\text{compon}(\mathcal{H}) \cup = \{Ch_i^{H,t}\}$
- 8: **end if**
- 9: **if** $i \leq k$ **then**
- 10: $Ch_i^{H,b} :=$ a new cluster with participants
 $Q_{i,1}^H, \dots, Q_{i,j_i}^H$ with the same structure as P_i
- 11: $\text{compon}(\mathcal{H}) \cup = \{Ch_i^{H,b}\}$
- 12: **end if**
- 13: $\text{label}_i :=$ empty label
- 14: **end if**
- 15: **end for**
- 16: **for** $i := 1$ to $k - 1$ **do**
- 17: $R_i^H := (Ch_i^{H,b}, Ch_{i+1}^{H,t}, E^{P_1, \dots, P_{i-1}}[P_i \rightarrow P_{i+1}], \text{label}_{i+1})$
- 18: $\text{compon}(\mathcal{H}) \cup = \{R_i^H\}$
- 19: **end for**
- 20: **if** $k > 0$ **then**
- 21: $R_k^H := (Ch_k^{H,b}, E^H, E^{P_1, \dots, P_{k-1}}[P_k \rightarrow E], l_E)$
- 22: $\text{compon}(\mathcal{H}) \cup = \{R_k^H\}$
- 23: **end if**
- 24: **for** $i := k + 1$ to n **do**
- 25: $R_i^H := (E^H, Ch_i^{H,t}, E^{P_1, \dots, P_k}[E \rightarrow P_i], \text{label}_{i+1})$
- 26: $\text{compon}(\mathcal{H}) \cup = \{R_i^H\}$
- 27: **end for**

Hierarchical structure construction

- 1: **for** $i := 1$ to k **do**
- 2: **for** $j := 1$ to j_i **do**
- 3: **if** $Q_{i,j}^H$ is a hierarchical entity type **then**
- 4: $\text{hier}(Q_{i,j}^H)_+ = (R_i^H)$
- 5: **else** $\{Q_{i,j}^H$ is a cluster of hierarchical entity
types $\}$
- 6: **for** each hierarchical entity type P^H being a
recursive participant in $Q_{i,j}^H$ **do**
- 7: $\text{hier}(P^H)_+ = (R_i^H)$
- 8: **end for**
- 9: **end if**
- 10: **end for**
- 11: **end for**
- 12: **for** $i := k + 1$ to n **do**
- 13: $\text{hier}(E^H)_+ = (R_i^H)$
- 14: **end for**

4.2 Join Operator

The join operator is used for joining two XSEM-H schemes to one XSEM-H schema. It joins one XSEM-H schema as a subschema of another XSEM-H schema. It allows to construct hierarchical representations of complex parts of XSEM-ER schemes from hierarchical representations of separate components.

The operator has the following parameters:

- an XSEM-ER schema \mathcal{ER}
- XSEM-H schemes \mathcal{H}_1 and \mathcal{H}_2 bounded with \mathcal{ER} , where names of the components of \mathcal{H}_1 and the names of the components of \mathcal{H}_2 are disjoint
- hierarchical entity types $\{E_{1,1}^H, \dots, E_{1,n}^H\}$ from \mathcal{H}_1 and $\{E_{2,1}^H, \dots, E_{2,n}^H\}$ from \mathcal{H}_2 , where:
 - $\{E_{2,1}^H, \dots, E_{2,n}^H\}$ are roots of \mathcal{H}_2 ,
 - $\forall 1 \leq i \leq n \text{ bound}(E_{1,i}^H) = \text{bound}(E_{2,i}^H)$
 - labels of hierarchical relationship types from $\text{hier}(E_{1,i}^H)$ must be disjoint with labels of hierarchical relationship types from $\text{hier}(E_{2,i}^H)$

- new hierarchical entity type names E_1^H, \dots, E_n^H

The algorithm for the operator is divided to two phases. In the first phase, the operator constructs an XSEM-H schema \mathcal{H} bounded with \mathcal{ER} containing the components of \mathcal{H}_1 and \mathcal{H}_2 with $E_{1,i}^H$ and $E_{2,i}^H$ joined to a new hierarchical entity type $E_i^H, \forall 1 \leq i \leq n$. In the second phase, for each new hierarchical entity type E^H the operator constructs a cluster of hierarchical relationship types in $hier(E^H)$ when there are all necessary components in $hier(E^H)$. It is described in detail in the formal description of the algorithm.

For example assume the XSEM-H schema at Figure 3.1. First, there were hierarchical representations \mathcal{H}^{Visit} composed of the hierarchical relationship types $R2, R3$, and $R4$, $\mathcal{H}^{Anamnesis}$ composed of $R5$, $\mathcal{H}^{Examination}$ composed of $R6$, and $\mathcal{H}^{VisitDesc}$ composed of $R7$. To compose the schema, the join operator was used. First \mathcal{H}^{Visit} and $\mathcal{H}^{Anamnesis}$ were joined, then the result was joined with $\mathcal{H}^{Examination}$, and at the end the result was joined with $\mathcal{H}^{VisitDesc}$. After the last join, the condition for the construction of the cluster of the relationship types $R5, R6$, and $R7$ was satisfied.

Join of XSEM-H Schemas

- 1: $\mathcal{H} := (compon(\mathcal{H}_1) \cup compon(\mathcal{H}_2), \mathcal{ER})$
- 2: **for** $\forall 1 \leq i \leq n$ **do**
- 3: $E_i^H := (hier(E_{1,i}^H) + hier(E_{2,i}^H), bound(E_{1,i}^H))$
- 4: **for** each component C of \mathcal{H} where $E_{1,i}^H$ or $E_{2,i}^H$, respectively, is a recursive participant of C **do**
- 5: replace $E_{1,i}^H$ and $E_{2,i}^H$, respectively, in C by E_i^H
- 6: **end for**
- 7: $compon(\mathcal{H}) := (compon(\mathcal{H}) \setminus \{E_{1,i}^H, E_{2,i}^H\}) \cup \{E_i^H\}$
- 8: **end for**

Construction of Clusters of Hierarchical Relationship Types

- 1: **for** $\forall 1 \leq i \leq n$ **do**
- 2: **if** $bound(E_i^H)$ is an entity type or a relationship type **then**
- 3: $\mathcal{P} := \{bound(E_i^H)\}$
- 4: **else**
- 5: $\mathcal{P} := \{parent(bound(E_i^H)), child(bound(E_i^H))\}$
- 6: **end if**
- 7: **for** each P in \mathcal{P} **do**
- 8: **for** each incoming cluster type C with the parent P where P_1, \dots, P_m are all the recursive participants of C being relationship types, weak entity types, and data node types **do**
- 9: **if** $\forall 1 \leq j \leq m$ $hier(E^H)$ contains a hierarchical relationship type R_j^H having its child participant bounded with P_j or with a hierarchical projection having P_j as a child or parent **then**
- 10: remove R_1^H, \dots, R_m^H from $hier(E_i^H)$
- 11: $C^H :=$ a new cluster of hierarchical relationship types R_1^H, \dots, R_m^H having the same structure as C
- 12: $hier(E_i^H) += C^H$
- 13: **end if**
- 14: **end for**
- 15: **end for**
- 16: **end for**

4.3 Contraction of Hierarchical Relationship Types

Complete full representations constructed by the operator for transformations of XSEM-ER relationship types and weak entity types are not suitable in every situation. Assume for example the relationship type $Employ$ from the schema at Figure 2.2. We can construct the complete full representation of $Employ$ shown at Figure 4.2 with hierarchical relationship types $R15$ bounded with $Employ[Physician \rightarrow Clinic + Department]$ and $R16$ bounded with $Employ^{Physician}[Clinic + Department \rightarrow Employ]$.

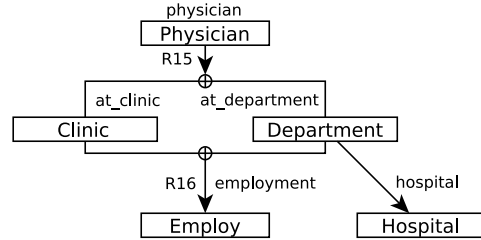


Figure 4.2: XSEM-H Schema

For example, the schema describes the structure of an XML document at Example 4.3.

```
<physician><name>Bill White</name>
  <at_department><name>Department A</name>
    <hospital><name>Hospital B</name>
  </hospital>
  <employment><since>2001-09-01</since>
</employment>
</at_department>
  <at_clinic><name>Clinic C</name>
    <employment><since>2005-03-01</since>
  </employment>
</at_clinic>
</physician>
```

Example 4.3: XML Described by an XSEM-H schema

However, there is a cardinality constraint $card(bound(R16), Clinic + Department) = (1, 1)$. It means that there can be only one employment of a physician at a clinic or department of a hospital. In such a case, it might be useful to contract the hierarchical relationship type $R16$ and join the hierarchical entity types bounded with $Clinic$ and $Department$ with the hierarchical entity type bounded with $Employ$. The result of the contraction is shown at Figure 3.1. For example, it describes the structure of an XML document at Example 4.4.

```
<physician><name>Bill White</name>
  <at_department><name>Department A</name>
    <hospital><name>Hospital B</name>
  </hospital>
  <since>2001-09-01</since>
</at_department>
  <at_clinic><name>Clinic C</name>
    <since>2005-03-01</since>
  </at_clinic>
</physician>
```

Example 4.4: XML Described by an XSEM-H schema

For such contractions we propose the operator for contractions of hierarchical relationship types. The parameters of the operator are the following:

- an XSEM-ER schema \mathcal{ER} and an XSEM-H schema \mathcal{H} bounded with \mathcal{ER}
- hierarchical relationship type R^H from \mathcal{H} , where:
 - R^H is bounded with ρ , where ρ is a hierarchical projection of a weak entity type or a relationship type R from \mathcal{ER} , $child(\rho) = R$, and $card(\rho, parent(\rho)) = (1, 1)$
 - $parent(R^H)$ is neither a hierarchical entity type bounded with a hierarchical projection nor a cluster of hierarchical entity types having any entity type bounded with a hierarchical projection and $child(R^H)$ is a hierarchical entity type bounded with R

If $parent(R^H)$ is a hierarchical entity type then the bottom of the resulting contracted full representation of R is this hierarchical entity type. If $parent(R^H)$ is a cluster of hierarchical entity types then the bottom of the resulting contracted full representation is this cluster.

Contract

- 1: $P^H := parent(R^H)$, $Q^H := child(R^H)$
- 2: **if** P^H is a hierarchical entity type **then**
- 3: $P_1^H := P^H$, $n := 1$
- 4: **else**
- 5: P_1^H, \dots, P_n^H are all the recursive participants of P^H being hierarchical entity types
- 6: **end if**
- 7: **for** each R'^H in $hier(Q^H)$ **do**
- 8: $parent(R'^H) := P^H$
- 9: **end for**
- 10: **for** $i := 1$ to n **do**
- 11: $hier(P_i^H)- = R^H$, $hier(P_i^H)+ = hier(Q^H)$
- 12: $bound(P_i^H) := R^{context(\rho)}[bound(P_i^H) \rightarrow R]$
- 13: **end for**
- 14: $compon(\mathcal{H}) \setminus = \{R^H\}$, $compon(\mathcal{H}) \setminus = \{Q^H\}$

5 Conclusions and Future Work

In this paper we proposed a new conceptual model for XML data called XSEM. The main contribution of the model is that the conceptual modeling process of XML data is divided to two levels. On the first level, an overall conceptual schema of the application domain is designed. On the second level, hierarchical organizations of parts of the overall conceptual schema are designed.

The approach integrates modeling of XML data to the process of modeling the whole application domain on the conceptual level. There is an interconnection between hierarchical organizations of data and the overall conceptual schema. The interconnection can be used to transform data in a hierarchical representation to a non-hierarchical internal representation described by the overall conceptual schema automatically.

We are going to propose algorithms for the translation of XSEM-H schemes to the logical XML level. Beside the grammar based XML schema languages such as XML Schema, we will study the usage of pattern based XML schema languages such as Schematron specified by ISO (2005) for the description of more complex integrity constraints. After this, we will create a prototype CASE tool for designing XSEM schemes.

We will study extensions of proposed key constraints of entity types. In XML, more extended types of keys than absolute keys of entity types and keys of weak entity types relative to its determinants can be

specified. It should be possible to specify a range of validity of a key of a given entity type in an XSEM-ER schema. For example, assume weak entity types *Section* and *Chapter* where *Chapter* is a determinant of *Section* and a strong entity type *Book* is a determinant of *Chapter*. A key *label* of *Section* would be relative to *Chapter*. However, we need to specify that the key is relative to *Book*, i.e. labels of sections are unique in the context of books and not in the context of chapters.

References

- Bird, L., Goodchild, A. & Halpin, T. A., (2000), Object Role Modelling and XML-Schema, Conceptual Modeling - ER 2000, Proceedings of the 19th International Conference on Conceptual Modeling, p.309-322, Salt Lake City, Utah, USA, October 9-12, 2000. Lecture Notes in Computer Science 1920 Springer 2000.
- Ceri, S., Fraternali, P., & Bongio, A. (2000), Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks, Volume 33, Numbers 1-6, p. 137-157, June 2000.
- Dobbie, G., Xiaoying, W., Ling, T.W. & Lee, M.L. (2000), ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. TR21/00, Department of Computer Science, National University of Singapore. December 2000.
- Duta, A. C., Barker, K. & Alhajj, R., (2004), ConVRel: relationship conversion to XML nested structures, Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), p.698-702, Nicosia, Cyprus, March 14-17, 2004.
- Fallside, D. C., Walmsley, P. XML Schema Part 0: Primer Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-0-20041028. October 2004.
- International Organization for Standardization, Information Technology Document Schema Definition Languages (DSDL) Part 3: Rule-based Validation Schematron. ISO/IEC 19757-3, February 2005.
- Necasky, M. (2006), Conceptual Modeling for XML: A Survey. Tech. Report No. 2006-3, Dep. of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, 2006, 54 p. <http://www.necasky.net/papers/tr2006.pdf>
- Pigozzo, P., Quintarelli E., (2005), An algorithm for generating XML Schemas from ER Schemas, in Proceedings of the Thirteenth Italian Symposium on Advanced Database Systems, SEBD 2005, p. 192-199, Brixen-Bressanone (near Bozen-Bolzano), Italy, June 19-22, 2005.
- Psaila, G. (2000), ERX: A Conceptual Model for XML Documents, in Proceedings of the 2000 ACM Symposium on Applied Computing, p. 898-903. Como, Italy, March 2000.
- Sengupta, A., Mohan, S. & Doshi, R. (2003), XER - Extensible Entity Relationship Modeling, in Proceedings of the XML 2003 Conference, p. 140-154. Philadelphia, USA, December 2003.
- Thalheim, B. (2000), Entity-Relationship Modeling: Foundations of Database Technology, Springer Verlag, 2000, Berlin, Germany.