Conceptual Modeling of IS-A Hierarchies for XML

Martin NECASKY and Jaroslav POKORY
Charles University, Czech Republic

Abstract. In this paper we briefly describe a new conceptual model for XML called XSEM. It is a combination of several approaches in the area. It divides the conceptual modeling process to conceptual and structural level. At the conceptual level, we design an overall conceptual schema of a domain independently on the required hierarchical representations of the data in XML documents. At the structural level, we design required hierarchical representations of the modeled data in different types of XML documents. In this paper, we further extend XSEM for modeling IS-A hierarchies. We also show how conceptual XSEM schemes can be represented at the logical level in XML schemes specified in the XML Schema language.

Keywords. XML, XML Schema, conceptual modeling, IS-A hierarchies

Introduction

XML has recently become an important format for data representation. It is widely applied as a database model, data exchange format, etc. This is mainly because of its simplicity, versatility, and platform independence. As XML data becomes more complex and mission critical the importance of its precise description at the logical and conceptual level grows as well. We can use XML schema languages such as XML Schema to describe XML data. Even though these languages provide powerful constructs for describing structure they are not however suitable for conceptual modeling of XML data. A conceptual model must allow to describe the data independently of any representation. From this point of view, XML schema languages are rather logical level languages then conceptual models. It is similar to the world of relational data where we describe structure of relations at the logical level with relational schemes but model them at the conceptual level with the E-R model or some of its variants. For XML data we therefore need an equivalent of the E-R model for its modeling at the conceptual level.

However, XML data model has some special features such as irregular structure, ordering, mixing structured and unstructured data, or hierarchical structure that are hard to model with the E-R model. Therefore, it is necessary to provide new approaches suitable to model these special features. We provide a survey of this area in [6] where we identify two groups of approaches.

The approaches in the first group, such as [2], [9], or [10], are based on extending E-R with new modeling constructs. However, modeling the required hi-
Hierarchical structure is problematic with these approaches. It is modeled either by a special type of hierarchical relationship types or by relationship types restricted to binary relationship types with a cardinality $1:N$. In [8] an algorithm for transforming general E-R schemes (containing $M:N$ and $n$-ary relationship types) to hierarchical XML schemes is proposed. Similarly, in [1] an algorithm for transforming ORM schemes to hierarchical XML schemes is proposed. The algorithms derive XML schemes from conceptual schemes automatically.

However, there is a common situation where we need to represent concepts such as books and their authors in two or more different hierarchical structures. The first structure can be a list of books and for each book a list of authors and the second structure can be a list of authors and for each author a list of his or her books. It depends on user requirements, which hierarchical representations of the concepts are suitable for our system. These hierarchical representations cannot be therefore neither determined automatically by the system nor specified by explicit hierarchical relationship types in the conceptual schema by the designer because the modeled semantics of the data would be hidden among a huge number of hierarchical relationship types that have no conceptual meaning.

The approaches in the second group emerge from hierarchical structure. Conceptual schemes are trees whose nodes are entity types and edges are relationship types. The most advanced model in this group was proposed in [4]. The model is called ORA-SS and allows to model generally $n$-ary relationship types which is not allowed by any other conceptual model in this group to our best knowledge. In the previous example with books we can model each required hierarchy with a separate hierarchical conceptual schema. However, while in E-R or its extensions we can model books by one entity type, in this approach books are represented by two different nodes, one in each schema. Consequently, the modeled semantics is hidden in the hierarchical structures.

The weak points of these approaches result from modeling the semantics and hierarchical structure of the data at the same level. Therefore, it is necessary to further extend recent conceptual models to be suitable for modeling XML data. In [5] we proposed a new conceptual model for XML data called XSEM. The basic idea of XSEM, that tries to precede the problems mentioned above, is to divide the modeling process to two levels called conceptual and structural level. While the semantics of the data is modeled at the conceptual level, the hierarchical representation of the data in XML documents is modeled at the structural level. Therefore, XSEM preserves the advantages of E-R, mainly its simplicity and clearness, and adds the ability to model how the data is represented in different hierarchical XML structures.

**Related Work and Contributions.** In this paper, we further extend XSEM with constructs for modeling IS-A hierarchies for XML. We also show how to represent XSEM schemes with IS-A hierarchies at the logical XML schema level. Authors of other conceptual models for XML, such as [4], [10] have also studied modeling IS-A hierarchies. However, their interest in modeling IS-A hierarchies was only marginal and insufficient. An exception is the model proposed in [3]. The authors also propose how to model IS-A hierarchies for XML at the conceptual level and how to represent them at the logical XML schema level. For modeling IS-A hierarchies at the conceptual level they extend their own conceptual model
called C-XML. However, C-XML does not divide the conceptual modeling process to the conceptual and structural level as XSEM. It models the semantics as well as structure of the XML documents in one schema. Therefore, we need to augment the approaches such as the one proposed in [3] to be applicable to our approach. It includes modeling IS-A hierarchies at the conceptual level and specification of their representation in different types of XML documents at the structural level. Such approach has not been studied in recent literature yet to our best knowledge.

[3] shows that the XML Schema language does not allow to describe XML structures that can be modeled at the conceptual level using some types of IS-A hierarchies. We discuss these cases later in the paper. To solve this problem, [3] extends XML Schema with new modeling constructs. In our approach, we show that the types of IS-A hierarchies that can not be expressed with XML Schema can be expressed as integrity constraints on XML documents expressed as XQuery expressions. Therefore, we do not need to extend XML Schema.

The paper is organized as follows. In Section 1, we provide a motivating example. Section 2 contains a brief description of XSEM. In Section 3, we extend the conceptual level of XSEM for modeling IS-A hierarchies. In Section 4, we show how to represent IS-A hierarchies at the structural level. In Section 5, we show how to represent conceptual XSEM schemes at the logical XML level in an XML schemes using the XML Schema language complemented with XQuery for some more advanced constraints. We conclude in Section 6.

1. Motivation

Assume a logistic company that ensures transports of goods between destinations. There are several types of transports depending on the mean of transport used. We consider ground and air transports. Figure 1 shows an XML document representing a ground transport and air transport to a target destination. The destination is represented by a root XML element target and the transports are represented by the child XML elements transport (the ground transport) and air-transport (the air transport).

The structure of the XML document can be described with an XML schema language like XML Schema. In this paper, we show how to model such XML document at the conceptual level and how to translate the conceptual description to an XML Schema representation. We are especially interested in modeling IS-A hierarchies. At the conceptual level, an IS-A hierarchy contains different types of concepts where some types are specializations of some other more general types. For example, transport is a general type of concepts in our example and there are some types specializing it such as ground transport or air transport. The specializing types have the same properties as the general type and can have some additional ones.

The XML document at Figure 1 demonstrates how transports organized in the IS-A hierarchy can be represented in XML. Ground transports are represented as XML elements transport whereas air transports are represented as XML elements air-transport. We assume that there is an XML schema that describes the structure of the XML document. There are the types of contents of transport
Figure 1. Demonstration of IS-A hierarchies in XML documents

and air-transport XML elements described in this XML schema. Both types have a common part which is composed of XML elements code, distance, and item (repeated once or more times). This common part can be described in a separate general type of content which is extended by the types for the XML elements transport and air-transport with some additional XML elements (to simplify the explanation, we do not consider XML attributes in the paper). Therefore, IS-A hierarchies at the XML schema level can be comprehended as references of extending types of content XML elements to previously defined types. In this paper, we are interested in how to specify the representation of conceptual level IS-A hierarchies at the XML schema level. There is not only one possible representation but several. For example, Figure 1 demonstrates just one of the possible XML representations of transports. However, there are other possibilities as well and it depends on user requirements which ones are suitable.

2. XSEM model

XSEM is a new conceptual model for XML. It allows to model special features of XML such as irregular structure, ordering, mixing structured and unstructured data, and hierarchical structure. On the other hand, it is a conceptual model. Therefore, it must abstract from the required XML representations and allow designers to concentrate purely on the semantics of the modeled data.

To achieve these two goals, XSEM is divided to two parts called XSEM-ER and XSEM-H. Similarly to other conceptual models, such as the well-known Entity-Relationship model (E-R), we use XSEM-ER to model real-world objects and relationships between them. At the conceptual level, it is not important how these concepts are represented in XML documents. An XSEM-ER schema describes purely the semantics of the modeled data. On the other hand, we require
the data modeled by the XSEM-ER schema to be represented in one or more types of XML documents. These types of XML documents are modeled at the structural level with XSEM-H.

2.1. XSEM-ER model

XSEM-ER is based on the E-R model. We use entity and relationship types for modeling real-world objects and relationships between them and some additional constructs for modeling the special features of XML data. These extending constructs are not important for the purposes of this paper. For their detailed description we refer to [5]. An example XSEM-ER schema is shown at Figure 2 modeling a part of a domain of a logistic company.

Entity types are used for modeling real-world objects. Each entity type is characterized by its name and zero or more attributes. We further distinguish strong and weak entity types. Strong entity types model objects, whose existence does not depend on other objects. A strong entity type is displayed by a box with the name in the box and attributes under the box. For example, there is a strong entity type Packet modeling transported packets.

A weak entity type models objects whose existence depends on other objects that are modeled by entity types that are assigned to the weak entity type as so called determinants. It is displayed by a box with an inner hexagon. The name is displayed in the box and attributes under the box. It is connected with each of its determinants by a solid arrow oriented to the determinant. For example, there is a weak entity type Transport. It models transports of packets. For each transport there must be a target destination, truck used for the transport, and driver who drove the transport. Therefore, Transport has three determinants - Destination, Truck, and Driver.

An entity type $E$ models a set of instances of $E$. This set is denoted $E^C$. For each attribute $A$ of $E$ with a domain $\text{dom}(A)$ there is a partial function that assigns to instances of $E$ values of $A$ from $\text{dom}(A)$. The value of $A$ assigned to an instance $e$ of $E$ is denoted $A : e$. Further, if $E$ is weak then each instance $e$ of $E$ has assigned an instance of each of the determinants of $E$. It specifies that the existence of $e$ depends on the existence of the assigned instances of the determinants. The instance of a determinant $D$ of $E$ assigned to $e$ is denoted $D : e$.

Secondly, there are relationship types. Relationship types are used for modeling relationships between real-world objects. Each relationship type is charac-
terized by its name, zero or more attributes, and two or more entity types that form the relationship type. These entity types are called participants of the relationship type. A relationship type is displayed by a hexagon with the name in the hexagon and attributes under the hexagon. It is connected with each of its participants by a solid arrow oriented to the participant. For example, there is a relationship type Item connecting the entity types Transport and Packet, i.e. the entity types are participants of Item. It models relationships between transports and packets, concretely that packets are items of transports. The example shows only relationship types with two participants. However, there can be relationship types with three or more participants as well.

Similarly to entity types, a relationship type $R$ models a set of instances of $R$. This set is denoted $R^C$. For each attribute $A$ of $R$ with a domain $\text{dom}(A)$ there is a partial function that assigns to instances of $R$ values of $A$ from $\text{dom}(A)$. Again, $A : r$ denotes the value of $A$ assigned to an instance $r$ of $R$. Moreover, each instance of $R$ has assigned an instance of each of the participants of $R$. $P : r$ denotes the instance of a participant $P$ of $R$ assigned to an instance $r$ of $R$.

2.2. XSEM-H model

An XSEM-ER schema models the semantics of data independently on representation of the data in XML documents. On the other hand, there can be several types of XML documents that represent different parts of the data in different hierarchical structures. To model these types of XML documents we use the XSEM-H model. For each required type of XML documents we design a separate XSEM-H schema. An XSEM-H schema targets one or more entity and relationship types from the XSEM-ER schema and describes how their instances are represented in the modeled XML documents. It does not model any additional semantics of the data. Therefore, XSEM-H schemes are called views on XSEM-ER schemes.

XSEM-ER is non-hierarchical because of non-hierarchical weak entity and relationship types. For each of these types there are several possibilities of its hierarchical representation. For example, assume the relationship type Item with the participants Transport and Packet. It is an $M:N$ relationship type. It means that each transport can have more items and each packet can be an item in more transports (packets are not transported directly to target destinations but through more destinations using more different transports). We can require a hierarchical structure where we have a list of transports and for each transport a list of its items, i.e. packets that are items of the transport. We can also require the reversed structure, i.e. to have a list of packets and for each packet a list of transports in which the packet was transported.

In both cases we represented the same relationship type Item but in different hierarchical structures. The situation is more complicated if we consider relationship or weak entity types with more than two participants or determinants, respectively, such as Transport for example. One of the required representations can be to have a list of transports and for each transport to have the target destination, truck, and driver. Another representation can be to have a list of target destinations and for each target destination to have a list of transports to this destination. For each transport we further want to have the truck and driver. There are also other possibilities of hierarchical representations of Transport.
2.2.1. Hierarchical projections

It is not enough to specify required hierarchical representations of weak entity and relationship types in such informal way. For their formal description we propose a formalism called *hierarchical projection*.

**Definition 2.1:**

A hierarchical projection of a relationship or weak entity type $T$ is an expression $T^{T_1,\ldots,T_k}[P \rightarrow Q]$ where $T_1,\ldots,T_k$, $P$, and $Q$ are participants or determinants, respectively, of $T$. $P$ or $Q$ (exclusively) can be $T$ itself. $P$ is called parent, $Q$ is called child, and the sequence $T_1,\ldots,T_k$ is called context.

To describe our previous hierarchical representations of *Item* and *Transport* we use hierarchical projections. We start with *Item*. Hierarchical projections

\[
\begin{align*}
\text{Item}[\text{Transport} \rightarrow \text{Packet}] & \quad (H1) \\
\text{Item}^\text{Transport}[\text{Packet} \rightarrow \text{Item}] & \quad (H2)
\end{align*}
\]

describe a hierarchical structure where we have a list of transports and for each transport we have a list of transported packets ($H1$). For each packet in the transport we further need the item relationship connecting the packet with the transport ($H2$). Similarly, we can describe the reversed representation by

\[
\begin{align*}
\text{Item}[\text{Packet} \rightarrow \text{Transport}] & \quad (H3) \\
\text{Item}^\text{Packet}[\text{Transport} \rightarrow \text{Item}] & \quad (H4)
\end{align*}
\]

describing a hierarchical structure where we have a list of packets and for each packet we have a list of transports in which the packet was transported ($H3$). For each transport in the packet we further need the item relationship connecting the transport with the packet ($H4$). Another hierarchical representation of *Item* is described by

\[
\begin{align*}
\text{Item}[\text{Item} \rightarrow \text{Transport}] & \quad (H5) \\
\text{Item}[\text{Item} \rightarrow \text{Packet}] & \quad (H6)
\end{align*}
\]

describing a hierarchical structure where we have a list of item relationships and for each item relationship we have the transport ($H5$) and packet ($H6$) connected by the relationship.

We also discussed hierarchical representations of *Transport*. It is a weak entity type with three determinants. Therefore, specification of its hierarchical representations is more complex. For example, the first representation, that we discussed previously, is described by hierarchical projections

\[
\begin{align*}
\text{Transport}[\text{Transport} \rightarrow \text{Destination}] & \quad (H7) \\
\text{Transport}[\text{Transport} \rightarrow \text{Truck}] & \quad (H8) \\
\text{Transport}[\text{Transport} \rightarrow \text{Driver}] & \quad (H9)
\end{align*}
\]

describing a hierarchical structure where we have a list of transports and for each transport we have its target destination ($H7$), truck ($H8$), and driver ($H9$). The second representation is described by
describing a hierarchical structure where we have a list of destinations and for each destination we have a list of transports to this destination \((H10)\). For each transport we further have its truck \((H11)\) and driver \((H12)\). Another representation is described by

\[
\begin{align*}
\text{Transport} & \rightarrow \text{Driver} \quad \quad \quad \text{(H13)} \\
\text{Transport}_{\text{Truck}} & \rightarrow \text{Driver} \quad \quad \quad \text{(H14)} \\
\text{Transport}_{\text{Truck,Driver}} & \rightarrow \text{Destination} \quad \quad \quad \text{(H15)}
\end{align*}
\]

describing a hierarchical structure where we have a list of trucks and for each truck we have a list of drivers who drove the truck \((H13)\). For each driver we have a list of transports driven by the driver \((H14)\). Because \((H14)\) has the context \textit{Truck} we consider also the superior truck for each driver and we therefore get only a list of transports driven by the driver in this truck. Because the driver can drive transports in more trucks we get a different list of transports for each of these trucks. Finally, for each transport we have the target destination \((H15)\).

### 2.2.2. XSEM-H views

Hierarchical projections allow to formally describe various hierarchical representations of weak entity and relationship types. However, it is not sufficient to provide designers only with hierarchical projections for modeling hierarchical structure because they require to work in a graphical environment. Therefore, we introduce so called \textit{XSEM-H views}. In this section, we describe only the basic form of XSEM-H views. There are also some additional constructs offered that allow to model more complex hierarchical structures. However, their description is out of the scope of this paper. We refer to [5] for their more detailed description.

**Definition 2.2:**

Let \(\mathcal{ER}\) be an XSEM-ER schema. An \textit{XSEM-H view} \(\mathcal{H}\) is a set of oriented trees. Each \textit{node} in \(\mathcal{H}\) represents an entity or relationship type from \(\mathcal{ER}\). A node can have assigned a label. Each \textit{edge} in \(\mathcal{H}\) represents a hierarchical projection of a weak entity or relationship type from \(\mathcal{ER}\). It is oriented from the \textit{parent node} to the \textit{child node}. \(\mathcal{H}\) can not have an arbitrary structure. Let \(e\) be an edge in \(\mathcal{H}\) with the parent \(N_p\) and child \(N_c\). Let \(e\) represent a hierarchical projection \(T^{T_1,\ldots,T_k}\) \([P \rightarrow Q]\). The following conditions must be satisfied:

1. \(N_p\) represents \(P\) and \(N_c\) represents \(Q\)
2. if \(k > 0\) then there is an edge having \(N_p\) as the child and representing a hierarchical projection \(T^{T_1,\ldots,T_{k-1}}[T_k \rightarrow P]\); if \(k = 0\) then there is no edge having \(N_p\) as the child and representing a hierarchical projection of \(T\)

These conditions ensure that the structure of the XSEM-H view corresponds to the structure described by the hierarchical projections represented by the edges.
A node can have assigned a cardinality constraint in a given edge. The cardinality constraints are formally specified on hierarchical projections. However, we do not describe cardinality constraints on hierarchical projections in a more detail in this paper. For a detailed information we refer to [5].

XSEM-H views are visualized in a similar way we visualize graphs. A node is displayed by a rectangle with the node name in the rectangle. If the node has a label than it is displayed above the node. An edge is displayed by a solid arrow oriented from the parent node to the child node.

Figure 3 shows two example XSEM-H views on the XSEM-ER schema at Figure 2. They model two different types of XML documents representing transports. Each node in the views is denoted by $U_{\text{Type}}$ where Type is the entity or relationship type represented by the node. Each edge is labeled with the hierarchical projection it represents. We use only the numbers of the hierarchical projections. They were specified in Section 2.2.1.

An XSEM-H view $\mathcal{H}$ specifies how instances of entity and relationship types represented by the nodes in $\mathcal{H}$ are represented in XML documents. The hierarchical structure is formally given by the hierarchical projections represented by the edges of $\mathcal{H}$. $\mathcal{H}$ itself only adds some supplemental information that is not provided by hierarchical projections but is necessary for the specification of the modeled structure of XML documents. This includes names of XML elements that are given by the labels assigned to the nodes in $\mathcal{H}$ and ordering on XML elements that is given by the ordering on the edges going from the corresponding node.

Formally, let $U_T$ be a node in $\mathcal{H}$ representing $T$. Let $t$ be an instance of $T$. $U_T$ specifies that $t$ is represented as a sequence of XML elements. The value of an attribute $A$ of $T$ assigned to $t$ is represented as an XML element with the name and type given by $A$. The XML elements representing the values of the attributes are ordered in the order given by $T$ and followed by representations of the instances of the entity and relationship types assigned to $t$ by the edges having $U_T$ as the parent. The ordering on the representations of the instances follows the ordering described by $U_T$. If $U_T$ has assigned a label, the XML representation of $t$ is enclosed into an XML element with the label as its name.
The following XML document has the structure described by the view (a). It is an XML representation of an instance of the entity type Packet as described by the view. The root node $U_{Packet}$ has a label packet and models the XML element packet. The attributes of the entity type Packet model the XML elements code and size. Further, there is the edge going from $U_{Packet}$ to $U_{Transport}$. $U_{Transport}$ has a label transport. It models the XML elements transport in packet. The attributes of Transport model the corresponding XML elements in transport. There are also XML elements position that are modeled by the attribute position of Item. Because $U_{Item}$ has not assigned a label the XML code modeled by $U_{Item}$ is not encapsulated in a separate XML element. Instead, it is included in the XML code modeled by the parent node $U_{Transport}$. Therefore, position are child XML elements of transport.

```
<packet>
  <code>DST281</code>
  <target>
    <code>PCKT83821</code>
    <gps>50N,14E</gps>
    <code>DST237</code>
  </target>
  <size>100x23x211</size>
</target>
<transport>
  <distance>872</distance>
</transport>
<code>T3821</code>
<transport>
  <distance>656</distance>
</transport>
<distance>872</distance>
<code>T4783</code>
<target>
  <position>18</position>
  <distance>245</distance>
</target>
<position>27</position>
</packet>
```

3. IS-A hierarchies in XSEM-ER

In the E-R model extended with IS-A hierarchies we can create an entity type $S$ that models some additional semantics to the semantics modeled by another entity type $G$. The additional semantics is modeled with attributes of $S$ and relationship types having $S$ as a participant. At the instance level, each instance $s$ of $S$ is also an instance of $G$. It means that $s$ has assigned not only values of the attributes of $S$ but also of $G$. We use this mechanism in XSEM-ER. For modeling IS-A hierarchies we use so called IS-A relationship types.

**Definition 3.1:**
Let $G$ and $S$ be two entity types. An IS-A relationship type is a pair $(G, S)_{IS-A}$ where $G$ and $S$ are called general and specializing entity type, respectively. We can also say that $G$ generalizes $S$ or that $S$ specializes $G$. At the instance level, the IS-A relationship type specifies that $S^C \subseteq G^C$.

Because each instance $s$ of $S$ is also an instance of $G$ it has assigned values of attributes of $G$ and also determinants of $G$, if $G$ has any. Therefore, $S$ is not described only with its own attributes and determinants. It is also implicitly described with the attributes and determinants of $G$. We call these attributes and determinants inherited attributes and determinants from $G$.

There can be another entity type $G'$ and an IS-A relationship type $(G', S)_{IS-A}$. In such case we say that there are multiple generalizations for $S$. It means that $S$ inherits attributes and determinants from $G$ and $G'$ as well.

We can also require several constraints to be held by general and specializing entity types. Assume entity types $G$, $S_1$, ..., $S_n$ and IS-A relationship types
$(G, S_1)_{IS-A}, \ldots, (G, S_n)_{IS-A}$. Without any constraints $\bigcup_{i=1}^{n} S^C_i \subseteq G^C$. Further, there can be $1 \leq i, j \leq n$ such that $S^C_i \cap S^C_j \neq \emptyset$.

Firstly, we can require $\bigcup_{i=1}^{n} S^C_i = G^C$ to be held. It means that each instance of $G$ must be also an instance of some of $S_1, \ldots, S_n$. This constraint is usually called union constraint. We specify it by denoting $G$ as abstract. Secondly, we can require $S^C_i \cap S^C_j = \emptyset$ for some $i, j$ where $i \neq j$, $1 \leq i, j \leq n$. We allow only a simpler constraint where we require $S^C_1, \ldots, S^C_n$ to be mutually exclusive, i.e. $\forall i, j$ where $i \neq j$ and $1 \leq i, j \leq n$ we require $S^C_i \cap S^C_j = \emptyset$. This constraint is usually called mutual-exclusion constraint. We specify it by denoting $G$ as exclusive.

We display IS-A relationship type by an empty arrow oriented from the specializing entity type to the general entity type. If the general entity type is denoted as abstract or exclusive then we mark the box with $A$ or $E$, respectively. Figure 4 shows an example of IS-A hierarchy. There is the weak entity type $Transport$ with a determinant $Destination$. The entity type models transports in general. We further distinguish air, ground, and water transports modeled by $AirTran$, $GroundTran$, and $WaterTran$, respectively, that specialize $Transport$. $Transport$ is an abstract and exclusive entity type. It means that a union constraint for $Transport$ and mutual-exclusion constraint for $AirTran$, $GroundTran$, and $WaterTran$ must hold. There are also other IS-A hierarchies in the schema. For example, the entity type $Contract$ models contracts with external business partners and water transports are contracts with external business partners. Therefore, $WaterTran$ specializes $Contract$. This is an example of multiple generalizations because $WaterTran$ specializes $Transport$ as well.

4. IS-A hierarchies in XSEM-H

In this section, we show how IS-A relationship types are represented in XSEM-H views. Suppose firstly that we construct an XSEM-H view that represents an entity type $E$ and there is one or more entity types generalizing $E$. To describe the hierarchical representation of $E$ we can specify hierarchical projections for the determinants of $E$ as well as its inherited determinants.

Assume for example the XSEM-H view at Figure 5. It represents the entity type $WaterTran$ that inherits determinants from $Transport$ and $Contract$. Its
hierarchical representation was constructed on the base of hierarchical projections WaterTran[WaterTran → Destination] and WaterTran[WaterTran → BusinessPartner].

![Diagram](image)

**Figure 5.** XSEM-H views

Secondly, suppose that we construct an XSEM-H view that represents an entity type $G$. We therefore construct an XSEM-H view $H_G$ from $G$ on the base of hierarchical projections of $G$. The resulting $H_G$ describes how instances of $G$ are represented in XML documents. Further suppose an entity type $S$ that specializes $G$. Because $S^C \subseteq G^C$, $H_G$ also describes the XML representation of instances of $S$. However, $S$ extends $G$ with some additional determinants whose hierarchical representation is not described with $H_G$. Therefore, we can require to construct an additional XSEM-H view $H_S$ that extends $H_G$. To denote that $H_S$ extends $H_G$ we use so called IS-A edges.

**Definition 4.1:**
Let $U_G$ and $U_S$ be two nodes in an XSEM-H view $H$. Let $U_S$ be a root in $H$. Let the nodes represent entity types $G$ and $S$, respectively. Let $(G, S)_{IS-A}$. An IS-A edge is a pair $(U_G, U_S)_{IS-A}$ where $U_G$ is called general node and $U_S$ is called specializing node of the IS-A edge. We also say that $U_G$ generalizes $U_S$ and $U_S$ specializes $U_G$.

The IS-A edge specifies that instances of $G$ are represented in XML documents according to the structure of the node $U_G$. If an instance of $G$ is also an instance of $S$ then it is represented in XML documents according to the structure of the node $U_G$ complemented with the structure of the node $U_S$. Moreover, if $U_S$ has a label than this label is used instead of the label of $U_G$.

The condition (2) from Definition 2.2 is not applied to the edges having $U_S$ as the parent before the completion. It must be satisfied after the completion. When we reformulate the condition, the following must hold. Let $e$ be an edge with the parent $U_S$ and representing a hierarchical projection $ST_1,\ldots,T_k[S \rightarrow Q]$. If $k > 0$ then there must be an edge having $U_G$ as the child and representing a hierarchical projection $GT_1,\ldots,T_k \rightarrow T_k \rightarrow G$ (because $S$ is generalized to $G$ we must replace $S$ with $G$). If $k = 0$ then there can not be an edge having $U_G$ as the child and representing a hierarchical projection of $G$.

In our graphical representation, an IS-A edge is displayed by an empty arrow oriented from the specializing node to the general node. Assume the example XSEM-H view at Figure 6. It shows how to model the XML representation of the IS-A edges from the XSEM-ER schema at Figure 4. This representation was demonstrated by the XML document at Figure 1. It contains an XSEM-H view $H_{Transport}$ representing the entity type $Transport$. It is composed of nodes
Figure 6. XSEM-H: IS-A hierarchies

$U_{Destination,1}$ and $U_{Transport}$. These nodes are connected by the edge that represents a hierarchical projection $Transport[Destination \rightarrow Transport] (H24)$.

Further, there are XSEM-H views $H_{AirTran}$ and $H_{GroundTran}$ representing the entity types $AirTran$ and $GroundTran$. $H_{AirTran}$ is composed of nodes $U_{AirTran}$, $U_{Pilot}$, and $U_{Aircraft}$ and two edges representing hierarchical projections $AirTran[Destination][AirTran \rightarrow Pilot] (H25)$ and $AirTran[Destination][AirTran \rightarrow Aircraft] (H26)$. Similarly, $H_{GroundTran}$ is composed of nodes $U_{GroundTran}$, $U_{Driver}$, and $U_{Truck}$ and two edges representing hierarchical projections $GroundTran[Destination][GroundTran \rightarrow Driver] (H27)$ and $GroundTran[Destination][GroundTran \rightarrow Truck] (H28)$.

The views $H_{AirTran}$ and $H_{GroundTran}$ extend the view $H_{Transport}$ which is denoted by the IS-A edges $(U_{Transport}, U_{AirTran})IS-A$ and $(U_{Transport}, U_{GroundTran})IS-A$. It means that instances of $Transport$ are represented in XML documents according to the structure of $U_{Transport}$. However, if an instance of $Transport$ is also an instance of $AirTran$ or $GroundTran$ then it is represented according to $U_{Transport}$ complemented with $U_{AirTran}$ or $U_{GroundTran}$, respectively. Representations of $Transport$ and $GroundTran$ instances are enclosed into XML elements $transport$ ($U_{GroundTran}$ does not have its own label) and representations of $AirTran$ instances are enclosed into XML elements $air-transport$. Because the edges representing the hierarchical projections (H25), (H26), (H27), and (H28) complement $H_{Transport}$ they must have $Destination$ as their context. Otherwise, the condition (2) given by Definition 2.2 would not be satisfied after the completion of $U_{Transport}$ with these edges.

5. Derivation of XML schemes

An XSEM-H view describes a given type of XML documents at the conceptual level. We need to derive an XML schema that describes this type at the logical level. To describe types of XML documents at logical level we use XML schema languages. In this section we show how to derive from an XSEM-H view a corresponding logical XML schema in the XML Schema language. For the derivation, we do not consider XML attributes but only XML elements. Firstly, we show a derivation of an XML schema from an XSEM-H view without IS-A edges. Secondly we consider IS-A edges.
5.1. XSEM-H views without IS-A edges

Assume a node $U$ in the XSEM-H view that represents an entity or relationship type $T$. From $U$ we derive a complex content `<xs:sequence><!-- components -->`<xs:sequence>` The complex content has assigned a unique name denoted $type_U$ (not in the XML schema directly, we will use the name later). The components of the complex content are derived from the attributes of $T$ and edges having $U$ as the parent:

1. for each attribute $A$ of $T$ with a domain $dom(A)$ we derive an element declaration `<xs:element name="A" type="dom(A)" />`
2. for each edge $E$ with the parent $U$ and child $V$ ($(min, max)$ is the cardinality of $V$ in $E$)
   - if $V$ has a label $l$ then we derive an element declaration `<xs:element name="l" type="type_V" minOccurs="min" maxOccurs="max"/>`
   - if $V$ has not a label then we derive a group declaration `<xs:group ref="type_V" minOccurs="min" maxOccurs="max"/>`

The components of the complex content definition are ordered in the order prescribed by $U$. Firstly, there are components derived from the attributes in the order prescribed by $T$. Secondly, there are components derived from the edges in the order prescribed by $U$.

The complex content derived from $U$ is included in the resulting XML schema in a form of complex type definition or group. If $U$ has a label then it describes XML elements named by this label. The content of these XML elements is described by the complex content derived from $U$. The complex content must be assigned to the XML elements as a complex type. Therefore, we put into the XML schema a complex type with the derived complex content and with $type_U$ as its name. If $U$ has not a label then it can not describe XML elements because we do not have a name for them. It describes only a group of XML elements derived from its attributes and edges. Therefore, we put into the XML schema a group with the derived complex content and with $type_U$ as id.

Finally, global element declarations for root nodes with labels are inserted into the resulting XML schema. If $U$ is a root node with a label $l$ then we derive a global element declaration `<xs:element name="l" type="type_U"/>`.

![Figure 7. XSEM-H view](image)

We demonstrate the derivation on example. Assume the XSEM-H view at Figure 7. For example, for $U_{Transport}$ we derive a complex type
We derived a complex type because \( U_{\text{Transport}} \) has the label \( transport \). The element declarations \( \text{code} \) and the commented ones are derived from the attributes of \( Transport \). The edge going to \( U_{\text{Item}} \) is represented as a reference to a group because \( U_{\text{Item}} \) has no label. The edge going to \( U_{\text{Destination}} \) is represented as an element declaration because \( U_{\text{Destination}} \) has a label. For \( U_{\text{Item}} \) we derive a group:

\[
<\text{xs:complexType name="Transport"} /> \\
<\text{xs:sequence}>
  <\text{xs:element name="code" type="xs:string"} />
  <!-- element declarations for the other attributes -->
  <\text{xs:group ref="Item"} />
  <\text{xs:element name="target" type="Destination"} />
</\text{xs:sequence}>
<\text{xs:complexType}>

For \( U_{\text{Packet}} \) and \( U_{\text{Destination}} \) we derive corresponding complex types in a similar way. Finally, we derive a global element declaration \( <\text{xs:element name="packet" type="Packet"} /> \) for the root node \( U_{\text{Packet}} \) because it has a label.

5.2. XSEM-H views with IS-A edges

In this section we show how to represent IS-A edges in derived XML schemes. As we showed in Section 5.1 there is a distinction between the representations of nodes with and without labels in XML schemes. To simplify the description of the representation of IS-A edges in XML schemes we however assume that each node has a label. It means that it is represented in the derived XML schema as a complex type. This label can be a label assigned directly to the node or a label inherited from another node. If a node has no label and is specialized by another nodes then it can be represented using \text{group} content model. However, it is technically complicated even though the idea is similar. Therefore, we think it is better to omit these details for the reader.

To describe the representation of IS-A edges in XML schemes we must distinguish several cases. Without a loss of generality assume nodes \( U_G, U_S, \) and \( U_{S'} \) representing entity types \( G, S, \) and \( S' \), respectively, and IS-A edges \( (U_G, U_S)_{IS-A} \) and \( (U_G, U_{S'})_{IS-A} \).

Suppose firstly that \( G \) is exclusive, i.e. \( S^C \cap S'^C = \emptyset \). It means that each instance of \( G \) that is not an instance of \( S \) nor \( S' \) is represented according to \( U_G \) and each instance of \( S \) or \( S' \) is represented according to \( U_S \) or \( U_{S'} \), respectively. Therefore, we derive complex contents from \( U_G, U_S, \) and \( U_{S'} \) by the basic algorithm in Section 5.1. Moreover, the complex contents derived from \( U_S \) and \( U_{S'} \) extend the complex content derived from \( U_G \). For this we use \text{extension} construction in the XML Schema language. It allows to extend an existing complex type (i.e. the one derived from \( U_G \)) to new complex types (i.e. the ones derived from \( U_S \) and \( U_{S'} \)).
Because $U_G$ has a label $l_{U_G}$ instances of $G$ are represented as XML elements named $l_{U_G}$ with the structure described by $type_{U_G}$. If $U_S$ has not its own label then it inherits $l_{U_G}$. It means that instances of $S$ are represented as XML elements named $l_{U_G}$ as well. However, their structure is described by $type_{U_S}$. Therefore, we need to declare XML elements named $l_{U_G}$, whose structure is described by $type_{U_G}$ or $type_{U_S}$. This is described in the derived XML schema already. We have the element declaration `<xs:element name="l_{U_G}" type="type_{U_G}" />` derived from $U_G$. The rules of the XML Schema language specify that each XML element described by this declaration has its content described by $type_{U_G}$ or any of the extended types, i.e. $type_{U_S}$ as well. If $U_S$ has its own label $l_{U_S}$ different from $l_{U_G}$ then instances of $S$ are represented as XML elements named $l_{U_S}$ and not $l_{U_G}$. It means that XML elements $l_{U_S}$ can appear wherever XML elements $l_{U_G}$ can appear. Therefore, we need to declare XML elements named $l_{U_G}$ with the structure described by $type_{U_G}$ and XML elements named $l_{U_S}$ with the structure described by $type_{U_S}$. Further, we must specify that XML elements $l_{U_S}$ can appear wherever XML elements $l_{U_G}$ can appear. The same must be done for $U_S'$.

If $U_G$ is a root node then XML elements $l_{U_G}$ are declared globally and we therefore declare globally XML elements $l_{U_G}$ and $l_{U_{G'}}$ as well. The result are global element declarations for $U_G$, $U_S$, and $U_S'$. If $U_G$ is not a root node then XML elements $l_{U_G}$ are declared locally in the complex content derived from the parent of $U_G$. We must specify that there can appear not only XML elements $l_{U_G}$ but also $l_{U_S}$ and $l_{U_{S'}}$. Therefore, we replace the original declaration of XML elements $l_{U_G}$ with

```
<xs:choice minOccurs="min" maxOccurs="max">
  <xs:element name="l_{U_G}" type="type_{U_G}" />
  <xs:element name="l_{U_S}" type="type_{U_S}" />
  <xs:element name="l_{U_{S'}}" type="type_{U_{S'}}" />
</xs:choice>
```

where $(min, max)$ is the cardinality of $U_G$ in the edge going to $U_G$.

Assume further that $G$ is not only exclusive but also abstract. It means that each instance of $G$ is also an instance of one of its specializations. If $S$ and $S'$ are all of its specializations then each instance of $G$ is represented according to $U_S$ or $U_{S'}$ and none is represented according to $U_G$. To denote this explicitly in the XML schema we can mark the complex type derived from $U_G$ as abstract using an XML Schema attribute `abstract` set to `true`. We also do not include the element declaration for $U_G$ in the XML schema. On the other hand, if there is moreover a specialization $S''$ of $G$ that is not represented in the XSEM-H view then instances of $S''$ are represented according to $U_G$. Therefore, we can not set the complex type derived from $G$ as abstract and the element declaration for $U_G$ must be included in the XML schema.

Secondly, let us discuss the situation where $G$ is not exclusive. It means that there can be an instance of $G$ that is an instance of $S$ and $S'$ at once. We suppose that $U_G$ has a label $l_{U_G}$. We start with $G$ being not abstract. If $U_S$ and $U_{S'}$ does not have their own labels they inherit the label from $U_G$. It means that each instance $g$ of $G$ is represented as an XML element $l_{U_G}$ according $U_G$. If $g$ is an instance of $S$, $S'$, or both ($G$ is not exclusive) then the structure of the XML element is complemented with $U_S$, $U_{S'}$, or both, respectively. It means that we
need a complex type for XML elements \( l_{U_G} \) that is composed of the complex content derived from \( U_G \) and optionally complemented with the complex contents derived from \( U_S \) and \( U_{S'} \), i.e.

\[
<\text{xs:complexType name="type}_{U_G}"> \\
<\text{xs:sequence}> \\
<!-- components for \( U_G --> \}<\text{xs:sequence minOccurs="0"> \\
<!-- components for \( U_S --> \}<\text{xs:sequence minOccurs="0"> \\
<!-- components for \( U_{S'} --> \}<\text{xs:sequence> \\
</xs:sequence> \\
</xs:complexType>
\]

If \( G \) is moreover abstract and there are no other specializing entity types of \( G \) the situation is more complicated. It means that each instance of \( G \) must be also an instance of \( S \) or \( S' \), or both (\( G \) is not exclusive). Therefore, each instance of \( G \) is represented as an XML element \( l_{U_G} \) according to \( U_G \) complemented with \( U_S, U_{S'} \), or both. However, the previously generated complex type allows an instance of \( G \) to be represented as an XML element \( l_{U_G} \) without these completions. We must therefore ensure that one or more of the sequences derived from the specializing nodes is repeated. In our case we get a choice of two variants: the first sub-sequence has \( \text{minOccur} = "1" \) and the second has \( \text{minOccur} = "0" \) or the first sub-sequence has \( \text{minOccur} = "0" \) and the second has \( \text{minOccur} = "1" \). This leads to a very complicated complex type if we consider tens or more specializing entity types. However, there is not a better solution using standardized XML Schema constructs. A better solution is to use a language that allows to describe such more advanced constraints in XML data. Such a language is for example XQuery.

We create the complex type \( \text{type}_{U_G} \) as in the previous case. For each sequence derived from a specialization of \( U_G \) we identify the first element declaration with \( \text{minOccur} = "n" \) where \( n \geq 1 \), if there is any. Therefore, we get a set of declarations of XML elements. Let \( e_1, \ldots, e_k \) be names of these XML elements. To check the required constraint we must check if some of these XML elements is present in the content of each XML element \( l_{U_G} \). The required constraint is then expressed by the following XQuery expression (\( \text{xpath}_{U_G} \) denotes an XPath expression targeting XML elements described by \( U_G \)):

\[
\text{for } \$e \text{ in } \text{xpath}_{U_G} \\
\text{return if count(\$e/\(e_1\))=0 AND ... AND count(\$e/\(e_k\)) = 0} \\
\text{then <validation-error/>} \\
\text{else <validation-ok />}
\]

Meanwhile, we supposed that \( U_S \) and \( U_{S'} \) inherit the label \( l_{U_G} \) from \( U_G \). Suppose now that they have their own labels \( l_{U_S} \) and \( l_{U_{S'}} \) that are not equal. It means that if \( g \) is an instance of \( S \) than it is represented as an XML element \( l_{U_S} \) according to \( U_S \). If it is an instance of \( S' \) than it is represented as an XML element \( l_{U_{S'}} \) according to \( U_{S'} \). The problem is that we require \( g \) to be represented as two different XML elements if it is an instance of both \( S \) and \( S' \) (it is possible
because \( G \) is not exclusive). Moreover, these XML elements have a common part described by \( U_G \). One possibility is to represent the common part as an XML element \( l_{UG} \) and the specializing parts as separate XML elements \( l_{US} \) and \( l_{U_S'} \). For this we need a mechanism of keys for entity types to interconnect these XML elements (they represent the same instance which must be explicitly denoted in the XML representation). However, modeling XML keys at the conceptual level is a complicated problem and we have no space to describe it in this paper. For a detailed description of modeling XML keys at the conceptual level with XSEM we refer to [7].

To solve this problem without a mechanism of keys we can represent each instance of \( U_G \) as an XML element \( l_{UG} \). If it is also an instance of \( S \) or \( S' \) we can include directly to the XML element \( l_{UG} \) an XML element \( l_{US} \) or \( l_{U_S'} \), respectively, containing the corresponding specializing part of the content. The resulting complex type describing XML elements \( l_{UG} \) is following:

\[
<\text{xs:complexType name="U}_G">\text{type}<\text{xs:sequence}>
  <!-- components for \( U_G \) -->
  <\text{xs:element name="U}_G" min\text{occurs}="0">\text{type}<\text{xs:complexType}>
    <!-- components for \( U_S \) -->
    <\text{xs:element}>
      <!-- and similarly for \( S' \) -->
    </\text{xs:element}>
  </\text{xs:sequence}>\text{type}<\text{xs:complexType}>
\]

If \( G \) is abstract we need to specify a similar constraint as in the previous case. Again we can express this constraint with a similar XQuery expression.

We demonstrate the basic ideas of the representation of IS-A hierarchies in XML schemes on examples. Suppose the XSEM-H view in Figure 6. We derive a complex type \( \text{Transport} \) from \( U_{\text{Transport}} \) by the basic algorithm. Then we derive complex types \( \text{AirTran} \) and \( \text{GroundTran} \) from \( U_{\text{AirTran}} \) and \( U_{\text{GroundTran}} \), respectively. Their complex contents are derived by the basic algorithm but the resulting complex types are extensions of \( \text{Transport} \). \( \text{AirTran} \) is derived as follows (\( \text{GroundTran} \) is derived similarly):

\[
<\text{xs:complexType name="AirTran"}>
  <\text{xs:complexContent}>
    <\text{xs:extension base="Transport"}>
      <\text{xs:sequence}>
        <!-- components derived from \( U_{\text{AirTran}} \) -->
      </\text{xs:sequence}>
    </\text{xs:extension}>
  </\text{xs:complexContent}>
</\text{complexType}>
\]

Finally, we derive declarations of XML elements \( \text{transport} \) and \( \text{air-transport} \). Because \( U_{\text{Transport}} \) has the parent \( U_{\text{Destination}} \), \( \text{transport} \) XML elements must
be declared locally in the complex content derived from $U_{Destination,1}$. Further, wherever a transport XML element can appear an air-transport XML element can appear as well. The complex type derived from $U_{Destination,1}$ therefore contains a choice of the declarations of these XML elements:

```xml
dx:
complexType
name= "Destination1"
sequence

element
declarations
for
the
attributes
choice

element
declaration
transport
name= "Transport"
type= "Transport"
choice

element
declaration
air-transport
name= "AirTran"
type= "AirTran"
choice
```

Even though the entity type $Transport$ is abstract the complex type $Transport$ is not denoted as abstract because there is also the entity type $WaterTran$ that is not represented in the XSEM-H view and its instances are therefore represented according to $U_{Transport}$, i.e. as XML elements $transport$ with the complex type $Transport$.

![Figure 8. XSEM-H views](image)

Finally, assume the XSEM-H view at Figure 8. It is rather a fragment of a larger XSEM-H view. We use it to demonstrate the representation of general entity types that are not exclusive. This is the case of the entity type $Employee$ specialized by $Driver$ and $Pilot$. We derive

```xml
<xs:element
name= "employee"
type= "Employee"
sequence

element
declaration
number
name= "number"
type= "string"

element
declaration
name
name= "name"
type= "string"

element
declaration
driver
name= "driver"
mminOccurs= "0"
sequence

element
declaration
drivenMiles
name= "drivenMiles"
type= "string"
```

If $Employee$ was abstract we would need to specify that each $employee$ XML element must contain $driver$ child XML element, $pilot$ child XML element, or both. This can be expressed by the following XQuery expression complementing the XML schema:
6. Conclusions

In this paper we described a conceptual model for XML called XSEM and we further extended it for modeling IS-A hierarchies. The proposed approach allows XML designers to abstract from complex constructions in XML schemes when representing IS-A hierarchies. It allows to start with modeling the semantics of the data at the conceptual level independently on its structural representation. After modeling the semantics, the designer can move to the structural level and specify how the modeled concepts including IS-A hierarchies are represented in several types of XML documents.

We showed how to represent XSEM schemes with IS-A hierarchies at the logical XML schema level using the XML Schema language. We showed that complicated cases of multiple generalizations and specializations unconstrained by union and mutual-exclusion constraints can be represented in XML schemes with standard constructions of XML Schema and XQuery.

References