Conceptual Model Based Normalization of XML Views*

Martin Nečaský

Department of Software Engineering, Faculty of Mathematics and Physics,
Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
martin.necasky@mff.cuni.cz

Abstract. As the popularity of XML as a format for data representation grows the need for storing XML data in an effective way grows as well. Recent research has provide us with effective solutions based on storing XML data into relational databases and with new technologies based on storing XML data in the native form. However, design of XML databases has not been studied sufficiently yet. In this paper, we suppose a set of XML schemes that describe XML representation of our data in several types of XML documents. We show that we can not usually store the data directly in this representation because it can contain redundancies. To design an optimal database schema we therefore need to locate these redundancies and eliminate them. We describe two types of redundancies in XML data in this paper and show how to utilize a conceptual schema of the XML schemes to locate such redundancies. We also show how to normalize the XML schemes to eliminate these redundancies.

Keywords: conceptual modeling, XML schema, normalization

1 Introduction

XML has become a popular format for data representation. Mainly it is because it is a variable format that is easy-to-use for a broad range of developers. Enterprises usually utilize several applications supporting different users for performing different business processes. Even though these applications share the same data (about customers, products, etc.), each of them requires the processed data to be represented in different forms suitable for the purposes of the application. XML proved itself as a suitable format for such various representations. For example, a sales reporting application for product managers represents customer's data in another type of XML documents than a web service for receiving and processing purchase orders from customers.

We need to store the data shared by the applications into a database and provide each application with the data represented in the required type or types

* This research was supported by the National programme of research (Information society project 1ET100300419) and by Grant Agency of Charles University (GAUK), grant number 204-10/257190

V. Snášel, K. Richta, J. Pokorný (Eds.): Dateso 2008, pp. 13–24, ISBN 978-80-248-1746-0.

of XML documents. We therefore comprehend these types of XML documents as *XML views* on the data stored in the database. These XML views are described by XML schemes. Given a set of such XML schemes the problem is how to design an optimal schema of the shared database. Even though we can use a native XML database to store the data in an XML representation, we can not usually store it directly as represented by the XML views. It is because the XML views can contain redundancies which means that the same data can be duplicated. Such a duplication means not only inefficient storage space usage but also problems when manipulating the data. We therefore need to identify these redundancies and eliminate them. For example, we can have an XML schema of an XML view for purchase orders where data about one product can be repeated in different purchase orders. This is a redundancy that should be eliminated. After the identification and elimination of all redundancies we get a set of normalized XML schemes. The situation is demonstrated in Figure 1. The idea is same as in the case of designing a relational database schema where we eliminate redundancies by modifying our schemes to meet so called *normal forms* such as 2NF, 3NF, or 4NF. This process is called *normalization*.

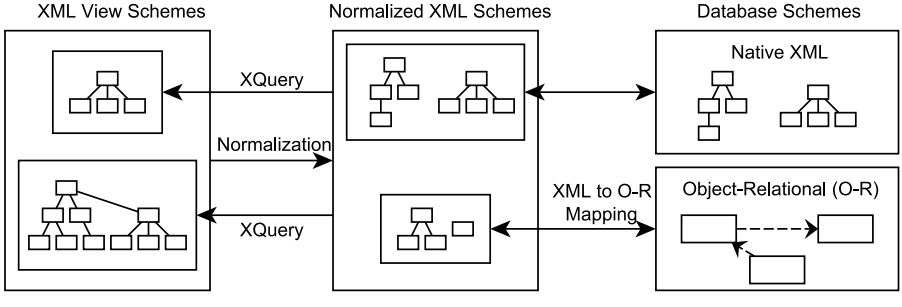

Fig. 1. Architecture

We can use the normalized XML schemes to design a schema of the database where we store the data shared by the applications. If our database is a native XML database (NXDB) we can directly use these normalized XML schemes as a NXDB schema. If our database is an object-relational database (ORDB) or a combination of ORDB and NXDB the normalized XML schemes are a good starting point to design the internal database schema. For example, we can map the normalized XML schemes into an ORDB schema. We can also combine both approaches and map structured parts of normalized XML schemes into an ORDB schema and use their unstructured parts or parts with a complex hierarchical structure as a schema of a NXDB. This situation is demonstrated in Figure 1.

In these cases, the database provides us the data in the form of XML documents with the structure given by the normalized XML schemes. However, we need to deliver the data to our applications in the form of the XML views. There-

fore, we moreover need a set of XQuery queries that transform the data between the normalized XML schemes representation and XML views. These queries can be derived automatically during the normalization process.

In this paper, we study normalization of a set of XML schemes as demonstrated in Figure 1. The other parts of the architecture displayed in the figure are out of the scope of this paper. Methods for mapping between XML and object-relational representations were studied for example in [5]. Derivation of the XQuery queries to reconstruct XML views is the subject of our further research. To normalize XML schemes we can apply the normal forms for relational data. Even though these normal forms should be considered when normalizing XML schemes we do not discuss them in this paper. We are interested in redundancies caused by hierarchical structure of XML schemes.

Related work. Several types of redundancies caused by hierarchical structure of XML schemes were also studied by other authors such as [1], [4]. Their results are based on functional dependencies in XML documents. In [3], authors show how to normalize XML schemes modeled in a richer model for XML data called ORA-SS. ORA-SS model is more a conceptual model then a logical XML model allowing to specify several integrity constraints for XML data. Authors show how to normalize XML schemes modeled as ORA-SS schemes using cardinality constraints. The advantage of this approach is that it is easier for the designer to specify cardinality constraints then discover functional dependencies in the hierarchical structure of the XML schema.

These approaches lead to good results when normalizing one XML schema. However, we need to normalize a set of XML schemes that can moreover represent the same data in different hierarchical structures. Discovering functional dependencies in such a set of XML schemes can be hard for the designer because he can be required to specify the same functional dependencies repeatedly for different XML schemes representing the same concept. Moreover, a concept represented in more different XML schemes also leads to redundancies as we show later in this paper. Such redundancies can not be identified and eliminated on the base of functional dependencies. Similar problems occur when we model XML schemes as ORA-SS schemes because each XML schema is modeled separately.

Contributions. In this paper we show how to normalize a set of XML schemes modeled at the conceptual level using a conceptual model for XML data called XSEM [6]. The advantage of this model is that the designer designs an overall non-hierarchical conceptual schema of the domain and derives the XML schemes of the required XML views from this overall conceptual schema. This allows to identify concepts that are represented in different XML views. Moreover, the designer is not required to specify functional dependencies or cardinality constraints repeatedly for different XML schemes. Instead, normalization is based on cardinality constraints specified in the overall conceptual schema where each cardinality constraint is specified only once.

Our approach can be applied in the systems where a large number of different XML views occurs and one or more databases to store the data in an effective way exits. In real systems, it is usually not required to fully normalize data

[2]. Following this requirement, it is not necessary to apply our approach to normalize the XML data fully to achieve better performance when reconstructing the views. The design of the normalized database schema is strongly influenced by the expected usage of data.

The paper is organized as follows. Section 1 is an introduction to the paper. In Section 2 we describe the XSEM model briefly. In Section 3 we describe two types of redundancies in XML data and we show how to eliminate them using the information from a conceptual schema. We conclude in Section 4.

2 XSEM Model

XSEM divides the conceptual modeling process to two levels. At the first level, we design an overall non-hierarchical conceptual schema of our domain using a part of XSEM called *XSEM-ER*. At the second level, we design hierarchical schemes as views on the XSEM-ER schema using a part of XSEM called *XSEM-H*. Each XSEM-H view schema describes an XML schema at the conceptual level. We briefly describe both parts of XSEM in this section. For a full and formal description of XSEM see [6].

2.1 XSEM-ER

XSEM-ER builds on an extension of the classical E-R model called HERM [8]. It allows to model real-world objects and relationships between them with entity types and relationship types and provides designers with extending constructs for modeling special XML features like irregular structure, ordering, and mixed content. In XSEM-ER, it is not important how the modeled data is organized in hierarchical XML documents. We show an example XSEM-ER schema modeling a small part of a business domain in Figure 2.

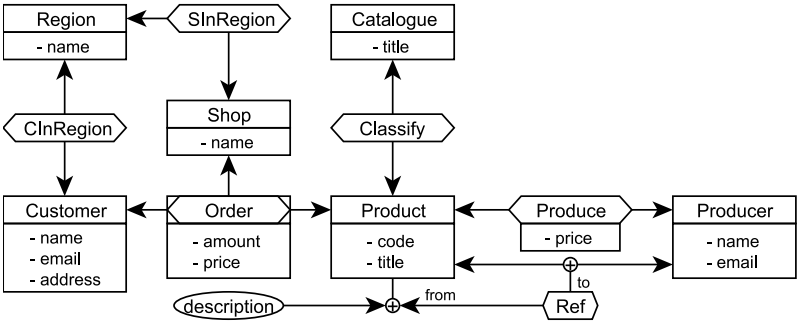

Fig. 2. XSEM-ER Schema for Business Company

The basic modeling constructs are *strong* and *weak entity type*, and *relationship type*. These constructs are known from the classical E-R model. Figure

2 shows strong entity types such as *Customer* and relationship types such as *Produce* with participants *Product* and *Producer*. It also shows weak entity types such as *Order* with determinants *Customer*, *Shop*, and *Product*.

There are two types of extending constructs. *Data node types* are used for modeling unstructured text parts of the data that can be mixed with structured parts. They are similar to attributes of entity or relationship types. However, they are not encapsulated directly in entity or relationship types but only assigned to them and grouped with another concepts in the schema. Data node types are displayed as ellipses. Figure 2 shows a data node type *description*. It models descriptions of products. We need to model that a description of a product can be mixed with references to other products and to producers. Therefore, we do not model description as an attribute of the entity type *Product* but as a data node type.

Cluster types are used for grouping different entity, relationship, and data node types. They are used to model irregular or mixed content at the conceptual level. We display cluster types as circles with inner +. There are *component* and *connection* cluster types. We use component cluster types for creating groups of two or more entity types. Such a group can then be assigned as a participant to a relationship type or determinant to a weak entity type. For example, there is a component cluster type composed of *Producer* and *Product*. This cluster type is assigned as a participant to a relationship type Ref. It models that references from products to other products and also producers.

Connection cluster types are used for creating groups of two or more concepts having the same entity type as a common participant or determinant. If there is a data node type in this group it models structured data mixed with unstructured data. For example, there is a connection cluster type composed of the data node type *description* and the relationship type Ref. It models that a description of a product is mixed with references to other products and producers.

An XSEM-ER schema does not specify how the data is organized in hierarchical XML documents. There is one or more possible hierarchical representations of each component of the schema. For example, we can represent instances of the weak entity type *Order* in the hierarchy where we have a list of orders and for each order we have the respective customer who made the order, ordered product, and shop where the order was made. We can also require another representation described as follows. We want a list of shops. For each shop we want a list of products ordered in the shop. For each such product we want a list of orders of the product made in the shop. Finally, for each such order we want the customer who made the order.

We need to describe such a hierarchical structure in a more formal way. For this we propose so called *hierarchical projections*. As an example, we show the following six hierarchical projections. The projections $(H1)$, $(H2)$, and $(H3)$ describe the former hierarchical representation of *Order*. The projections $(H4)$, $(H5)$, and $(H6)$ describe the other.

$$Order[Order \rightarrow Customer] \quad (H1)$$
$$Order[Order \rightarrow Product] \quad (H2)$$
$$Order[Order \rightarrow Shop] \quad (H3)$$
$$Order[Shop \rightarrow Product] \quad (H4)$$
$$Order^{Shop}[Product \rightarrow Order] \quad (H5)$$
$$Order^{Shop,Product}[Order \rightarrow Customer] \quad (H6)$$

Formally, a hierarchical projection h of an entity or relationship type T is an expression $T^{E^1,\ldots,E^k}[P \rightarrow Q]$ where $E^1, \ldots, E^k, P, Q$ are determinants or participants, respectively, of T. It specifies a hierarchy where P (called *parent*) is superior to Q (called *child*). The sequence $E^1, \ldots, E^k$ is called *context* and specifies the context in which the projection is considered. For example, $(H6)$ specifies a hierarchy where $Order$ is superior to $Customer$ in the context of $Shop$ and $Product$.

We also extend the notion of *cardinality constraints* for hierarchical projections. For the hierarchical projection h of T, we can specify a cardinality constraint for the parent or child, i.e. $card(h, P) = (m, n)$ or $card(h, Q) = (m, n)$, respectively. It means that for instances of the components from the context of h an instance of P (or Q, respectively) can appear in T with m up to n different instances of Q (or P, respectively). For example, a cardinality constraint $card(H6, Customer) = (0, *)$ specifies that for a given shop and product a customer can make an arbitrary number of orders of the product in the shop. A cardinality constraint $card(H6, Customer) = (0, 1)$ specifies that for a given shop and product a customer can make zero or one order of the product in the shop but not more.

2.2 XSEM-H

An XSEM-H schema models one type of XML documents. It is a view on a part of the XSEM-ER schema and specifies how the data described by this part of the XSEM-ER schema is represented in the modeled type of XML documents. It does not describe any further semantics of the data. We can derive several XSEM-H view schemes from the same part of the XSEM-ER schema. Therefore, they are not derived automatically but by the designer according to the required structure of the XML documents. Figure 3 shows three XSEM-H view schemes. For example, $CatalogueView$ describes the structure of XML documents with catalogue data.

An XSEM-H view schema is a set of trees with labeled oriented edges. Nodes in the view schema represent entity types, relationship types, and data node types. For clearness, we denote the nodes by $U_{T,n}$ where T is the type represented by the node and n is the counter for the nodes in the view schema representing T. For example, $OrderView$ contains a node U_{Order} representing the weak entity type $Order$ from the XSEM-ER schema. Edges in XSEM-H view schemes represent hierarchical projections of the types represented by the nodes. For example, the edge going from U_{Order} to $U_{Customer}$ in $OrderView$ represents the hierarchical projection $H1$, i.e. $Order[Order \rightarrow Customer]$. Nodes can have assigned labels displayed above the nodes. These labels are names of elements that are used to represent the nodes in the modeled type of XML documents. For example, U_{Order} in $OrderView$ has assigned a label *order*. It means that orders are represented in the modeled type of XML documents by elements `order`.

There are also several types of auxiliary nodes in XSEM-H view schemes. There are *cluster nodes* representing cluster types from the XSEM-ER schema.

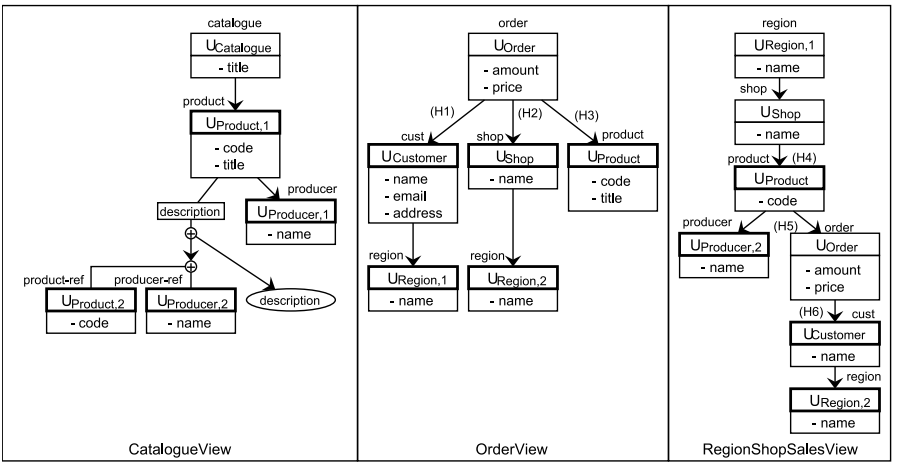

Fig. 3. XSEM-H view schemes for business company

They are displayed in the same way, i.e. as circles with an inner '+' symbol. Further there are so called *containers* that represent XML elements that group two or more different concepts but not have any equivalent at the conceptual level in the XSEM-ER schema. A container is displayed as a narrow rectangle with its name in the rectangle. For example, Figure 3 shows a container *description* assigned to the node $U_{Product,1}$. For a more detailed description of modeling constructs of XSEM-H, we refer to [6].

Each XSEM-H view schema models an XML schema at the conceptual level. This XML schema can be derived from the XSEM-H view schema automatically represented in a selected XML schema language. The derivation is straightforward. However, we do not discuss it in this paper because of the lack of the space.

3 Normalization

A set of XSEM-H view schemes can lead to redundancies when we represent our data in the respective types of XML documents. Normalization means to transform this set of XSEM-H view schemes to another set of XSEM-H view schemes that describe the same data but do not lead to redundancies. In this section, we show two types of redundancies and how to normalize XSEM view schemes that lead to such redundancies. Our goal is to modify the structure of the XSEM-H view schemes as little as possible during the normalization. We call the normalized XSEM-H view schemes *XSEM-H repository schemes* to distinguish them from the original ones.

Local redundancies. The first type of redundancies is caused by hierarchical projections with the maximal cardinality of their child greater then 1. In such case,

an instance of the child can be assigned to more different instances of the parent and therefore repeated in the respective hierarchical structure. Assume for example $SalesView$ in Figure 3. There is an edge going from $U_{Region,1}$ to U_{Shop} that represents a hierarchical projection $SInReg[Region \rightarrow Shop]$. The cardinality of $Shop$ in the projection is $(1, 1)$. Therefore, an instance of $Shop$ is assigned to one and only one instance of $Region$ and is therefore not repeated in the respective hierarchical structure. On the other hand, the edge going from U_{Shop} to $U_{Product}$ represents a hierarchical projection $Order[Shop \rightarrow Product]$ and the cardinality of $Product$ in this projection is $(0, *)$. It means that an instance of $Product$ can be repeated in zero or more instances of $Shop$.

On the base of this observation we define the first type of redundancies called *local redundancies*.

Definition 1. *Let U be a non-root node in an XSEM-H view schema. Let U represent a type P. Let e be an edge going to U and representing a hierarchical projection $T^{T_1,\ldots,T_{k-1}}[T_k \rightarrow P]$. Let the maximal cardinality of P in the hierarchical projection be greater than* 1. *If U represents one or more attributes of T or there is an edge going from U and representing a hierarchical projection with an empty context then we say that U* leads to local redundancies.

Assume that a node U, that represents a type P, leads to local redundancies. To eliminate these redundancies we normalize U by dividing it to two parts. The first part is called *context-independent* part of U and is composed of the attributes represented by U and edges going from U and representing hierarchical projections with an empty context. The second part is called *context-dependent* part of U and is composed of the edges going from U and representing hierarchical projections with a non-empty context. If an instance p of P is repeated at the location specified by U its content corresponding to the context-independent part of U is repeated as well. The content of p corresponding to the context-dependent part of U is different for each representation of p because it depends on the context. The normalization of U means to move its context-independent part to another node V that represents P as well but does not lead to local redundancies. The instance p of P is repeated at the location specified by V only once and its content corresponding to the context-independent part of U is therefore repeated only once as well. If such a node V does not exist we create a new XSEM-H repository schema and create V as its root node. The created node does not lead local redundancies because it is a root node. We call V *storage node* for P.

To reconstruct the original view we must be able to join U with its context-independent part moved to the storage node V. Joins are usually performed using keys and foreign keys. However, we did not show how to model keys in XSEM-ER schemes in this paper. We discussed this problem in [7]. The proposed keys can be used for modeling general XML keys that can be rather complex. This type of general keys is not however suitable for our purposes in this paper. Instead, we use a much simpler mechanism of artificial keys. We add an artificial key attribute *oid* to V and foreign key *oid* to U referencing *oid* in V. When reconstructing the original U, we join the normalized U with V using this pair.

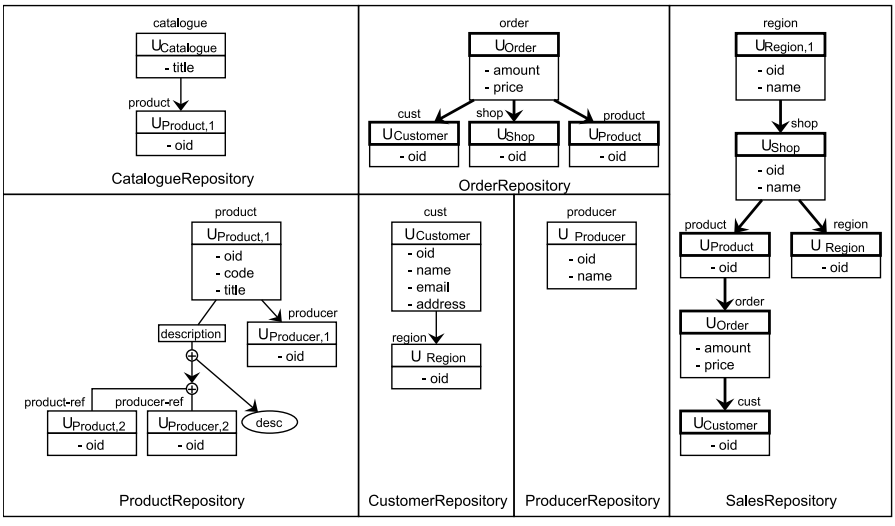

Fig. 4. XSEM-H repository schemes without nodes leading to local redundancies

Figure 3 shows nodes that lead to local redundancies in a bold line. Figure 4 shows the result of their elimination. For example, the node $U_{Product,1}$ in $CatalogueView$ leads to local redundancies. It is because the edge going to the node represents a hierarchical projection $Classify[Category \rightarrow Product]$ and the maximal cardinality of $Product$ in this projection is $*$. Normalization of the node means to move its context-independent content to the storage node for $Product$. However, all nodes in the XSEM-H view schemes that represent $Product$ lead to local redundancies and the storage node for $Product$ must be created. We therefore create a new XSEM-H repository schema $ProductRepository$ with a root node $U_{Product,1}$ representing $Product$. This node is a new storage node for $Product$. We move to this node all the attributes represented by $U_{Product,1}$ in $CatalogueView$ and all the edges representing hierarchical projections without a context, including clusters of edges and containers that contain these edges. Moreover, we add an artificial key attribute oid to $U_{Product,1}$ in $ProductRepository$ and foreign key attribute oid to $U_{Product,1}$ in $Catalogue-Repository$. The result of the normalization is that we store $Product$ instances according to $U_{Product,1}$ in $ProductRepository$. At the location specified by $U_{Product,1}$ in $CatalogueRepository$ we do not repeat whole $Product$ instances but only their values of the artificial foreign key oid. To reconstruct the original view we use this foreign key.

The other nodes representing $Product$ in the XSEM-H view schemes lead to local redundancies as well and are therefore normalized in the same way. We move the context-independent contents of these nodes to the previously created storage node $U_{Product,1}$ in $ProductRepository$. For the node $U_{Product}$ in $OrderView$ we move all its attributes. For the node $U_{Product}$ in $SalesView$

we move all its attributes and the edge going to $U_{Producer,2}$. The edge going to U_{Order} is in the context-dependent part of $U_{Product}$ and is therefore not moved.

Assume further U_{Shop} in $OrderView$ that also leads to local redundancies. To normalize it we do not need to create a storage node for *Shop* as in the previous case with *Product*. There is the node U_{Shop} in $SalesView$ that does not lead to local redundancies and each *Shop* instance is represented at this location. We can therefore use it as the storage node for *Shop* and we move here the context-independent content of U_{Shop} in $OrderView$.

Structural redundancies. The second type of redundancies in XSEM-H view schemes we discuss in this paper is caused by representing an entity or relationship type P at two or more different locations in XSEM-H view schemes. In such a case an instance of P can be repeated at two different locations in the respective XML representations. Assume for example the weak entity type *Order*. It is represented in $SalesView$ and $CustomerView$ as well. After the elimination of local redundancies we still have *Order* represented in two XSEM-H repository schemes $SalesRepository$ and $CustomerRepository$. It means that we represent an instance of *Order* twice in the respective XML representations. Once according to the former repository and once according to the other. We call this type of redundancy *structural redundancy*.

Definition 2. *We say that an entity or relationship type* leads to a structural redundancies *if it is represented at two or more different locations in XSEM-H view schemes.*

Assume that an entity or relationship type P leads to structural redundancies. To eliminate these redundancies we select one of its representations as *primary* and the others as *secondary*. We will use the primary representation for representing instances of P and the secondary representations will be reconstructed by XQuery queries. The selection of the primary representation is made by the designer. He can decide on the base of the usage of the representations. The most used representation should be selected as the primary one. The reader could argue that some more explicit guidelines to select the primary representation should be given. These guidelines could be based on statistics of the usage of the original views combined with the price of the reconstruction of the secondary representations. However, these guidelines overcome the scope of this paper.

Figure 4 shows XSEM-H repository schemes where nodes leading to local redundancies were normalized. However, there are several nodes that lead to structural redundancies and we need to normalize them. Figure 5 shows the resulting set of XSEM-H repository schemes after their normalization. Firstly, the relationship type $SInRegion$ is represented in $ShopRepository$ twice. The former representation is composed of the nodes $U_{Region,1}$ and U_{Shop} and the edge connecting them. The other representation is composed of U_{Shop} and $U_{Region,2}$ and the edge connecting them. It means that $SInRegion$ leads to structural redundancies. In other words each $SInRegion$ instance is represented in two different locations. To eliminate this structural redundancies we must select one

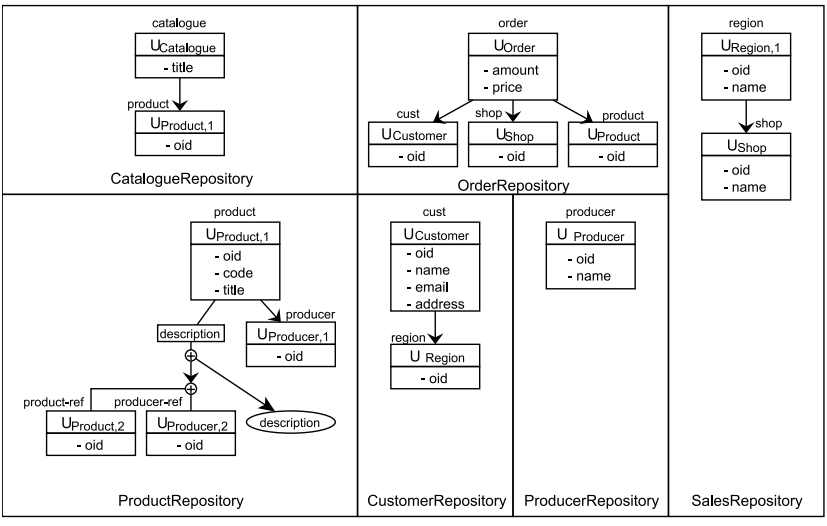

Fig. 5. XSEM-H repository schemes without nodes leading to local nor structural redundancies

of the representation as primary. We select the former representation as primary because it will be used more frequently then the other and its reconstruction would be therefore more expensive. The other representation is secondary and therefore not included in the resulting repository.

Another structural redundancy is the weak entity type *Order*. It is represented once in *OrderRepository* and once in *SalesRepository*. In both repositories the representation is composed of nodes U_{Order}, $U_{Customer}$, U_{Shop}, and $U_{Product}$. We select the representation in *OrderRepository* as primary. The other representation is not included in the resulting repository. The removed secondary representations of *SInRegion* and *Order* are not represented in the normalized XSEM-H repository schemes and must be therefore reconstructed from them by XQuery queries. These queries can be derived automatically. However, such derivation is out of the scope of this paper.

4 Conclusion

In this paper we showed how to model a set of XML schemes at the conceptual level using a conceptual model for XML data called XSEM. This model allows to model data at two levels. At the first level, an overall conceptual schema of the data is designed using a part of XSEM called XSEM-ER. At the second level, a conceptual schema modeling a given XML schema is derived from the XSEM-ER schema using a part of XSEM called XSEM-H. We further showed how to normalize a given set of XML schemes modeled by XSEM-H schemes. We described two types of redundancies caused by hierarchical structure of the XML

schemes, namely local and structural redundancies and showed how to eliminate these redundancies by normalization of the XSEM-H schemes. We also showed that these normalized XML schemes can be used to design a database schema suitable to store our data without redundancies.

References

1. Arenas, M., Libkin, L.: A Normal Form for XML Documents, *in* ACM Transactions on Database Systems (TODS), 29 (2004), pp. 195-232.
2. Balmin, A., Papakonstantinou, Y.: Storing and Querying XML Data Using Denormalized Relational Databases, *in* The VLDB Journal, 14(1), pp. 30-49, 2005.
3. Dobbie, G., Xiaoying, W., Ling, T.W., Lee, M.L.: ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. TR21/00, Department of Computer Science, National University of Singapore. December 2000.
4. Lee, M. L., Ling, T. W., Low, W. L.: Designing Functional Dependencies for XML, *in* Proceedings of the 8th Conference on Extending Database Technology (EDBT), Prague, March 2002, pp. 124-141.
5. Mlynkova, I., Pokorny, J.: XML in the World of (Object-)Relational Database Systems. *in* Proceedings of the 13th International Conference on Information Systems Development, Vilnius, Lithuania. Springer Science+Business Media, Inc., 2005. pp. 63-76,
6. Necasky, M.: XSEM - A Conceptual Model for XML. *in* Proceedings of the 4th Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Australia. CRPIT 67. 2007. pp. 37-48.
7. Necasky, M., Pokorny, J. Extending E-R for Modelling XML Keys. *in* Proceedings of the 2nd International Conference on Digital Information Management. IEEE Computer Society. Lyon, France, 2007, pp. 236-241.
8. Thalheim, B.: Entity-Relationship Modeling: Foundations of Database Technology. Springer Verlag, Berlin, Germany. 2000.