Designing Semi-structured Databases: A Conceptual Approach

Mong Li Lee, Sin Yeung Lee, Tok Wang Ling, Gillian Dobbie
School of Computing
National University of Singapore
{leeml, jlee, lingtw, dobbie}@comp.nus.edu.sg

Leonid A. Kalinichenko
Institute for Problems of Informatics
Russian Academy of Sciences
leonidk@synth.ipi.ac.ru

Abstract

Semi-structured data has become prevalent with the growth of the Internet. The data is usually stored in a traditional database system or in a specialized repository. While many information providers have presented their databases on the web as semi-structured data, other information providers are developing repositories for new application. One such application is e-commerce, which is emerging as a major web-supported application assisting business transactions between multiple parties via the network and involving large amounts of data. Designing a "good" semi-structured database is increasingly crucial to prevent data redundancy, inconsistency and updating anomalies. In this paper, we propose a conceptual approach to design semi-structured databases. A conceptual layer which is based on the popular Entity-Relationship (ER) model is employed to remove anomalies and redundancies at the semantic level. An algorithm to map an ER diagram involving composite attributes weak entity types, recursive, n-ary and ISA relationship sets, and aggregations to a semi-structured schema graph (S3-Graph) used to represent semi-structured data is given. Our study reveals similarities between the S3-Graph and the hierarchical model and nested relations in that all have limitations in modeling situations with nonhierarchical relationships given their tree-like structures.

1 Introduction

XML - the eXtensible Markup Language, has expanded beyond its document markup origins to become the new standard for data representation and exchange on the World-Wide Web [3]. The industry is racing to provide infrastructure for XML technologies, e.g. information interchange using XML, effective querying of XML sources, integration of XML data, and the storage and management of XML data. We address the storage and management of XML data. In fact, XML data is as a special case of semi-structured data, and we broaden our scope to semi-structured in this paper. Consider, for example, an electronic commerce (e-commerce) application, which is web-supported and assists business transactions between multiple parties via the network. Building an e-commerce application involves designing and maintaining databases (or repositories) of data including product catalogs, customer and vendor information and business-to-business transactions. The data can be stored in a relational database, an object-relational database, an object-oriented database or a specialized semi-structured repository. Currently application builders must define and create objects and relationships in the underlying databases, as the details of the schema are not expressed in the data, and hence cannot be extracted from the data source. As with traditional databases, poorly designed databases contain redundant data, leading to undesirable anomalies. The semi-structured data or XML that specifies the data, may be considered as one particular view of the data stored in the database. The presentations described using style sheets (XSL) are also views of the underlying data. If an object-relational database, like Oracle 8 is the underlying database, a design methodology that seamlessly maps from the objects and relations in the database, to the hierarchical elements and attributes in semi-structured data is required.

In this paper, we describe research that forms the basis of such a methodology. The methodology that we propose for building an application with semi-structured data, like an e-commerce application consists of three steps:

1. model the underlying database using ER diagrams [6],
2. normalize the ER diagrams [11],
3. model the views on the data using S3-graphs [10].

We have presented the methodology as a methodology for applications that are being built from scratch. It is also applicable to applications built on top of existing databases. Introducing this normalized layer to semi-structured database design has the following advantages:

1. anomalies and redundancies can be removed at the semantic level,
2. customized XML views can be generated from the normalized model, and
3. data can be stored in a relational database with controlled or no redundancy.

The contribution of this paper is an algorithm that maps the ER diagrams to semi-structured schema graphs (S3-Graphs), forming a seamless mapping between how the stored data is modeled and the semi-structured views of the data. Our study also reveals similarites between the S3-Graph and the hierarchical model and nested relations in that all have limitations in modeling situations with nonhierarchical relationships given their tree-like structures.

The rest of the paper is organized as follows. Section 2 reviews some background concepts, including XML, the S3-graph and the ER approach. Section 3 illustrates how the ER approach can be used to design a semi-structured database and gives a comprehensive algorithm to translate an ER diagram into a normal form semi-structured schema graph. Finally, we conclude in Section 4.

2 Preliminaries

In this section, we provide a brief overview of XML and demonstrate how an XML document can be represented graphically using the S3-graph. We also illustrate how an S3-graph that has not been properly designed can contain anomalies. We review some basics of the ER approach and show how it can be used to model semi-structured data.

2.1 XML Basics

The eXtensible Markup Language, XML, provides a hierarchical structure for exchanging information on the World Wide Web An XML document consists of nested element structures, starting with a root element. Element data can be presented in the form of attributes or sub-elements. Figure 1 shows an example XML document that contains information about students and the courses they are enrolled in. Each Course element has subelements Code and Title. Each Student element has subelements Name and Course which has a id attribute referencing some Course element. Course is further nested to contain Grade and Tutor information. Each Tutor element has subelements TName, Office and Feedback. Further information on XML can be found in [3].

XML has characteristics which are very similar to semi-structured data: self-describing, deeply nested or even cyclic, and irregular. Graph based complex object models such as Object Exchange Model (OEM) [15], Document Object Model (DOM) [7], Araneus Data Model (ADM) [14] and semi-structured graph [10] provide a natural framework for describing semi-structured XML repositories and their DTDs. Figure 2 shows the corresponding semi-structured data graph for the XML document in Figure 1. The semi-structured data graph is basically a labeled graph in which vertices correspond to objects and edges represent the object-subobject relationship. Each edge has a label describing the precise nature of the relationship. Each object has an object identifier (oid) and a value. The value is either atomic or complex, that is, a set of object references denoted as a set of (label, oid) pairs. More details of the semi-structured graph will be given in the next section.

2.2 Semi-Structured Graph

Data modeling people have acknowledged the fact that if the database attributes are fixed in structure, the modeling power of the data model is greatly reduced. With the introduction of XML, semi-structured data becomes widespread. In this section, we briefly review some of the concepts of a semi-structured data graph and a semi-structured schema graph (S3-Graph) defined in [10]. We will also discuss some of the anomalies that may appear in this model and how restructuring by decomposition can remove them.

A *Semi-Structured Schema Graph (S3-Graph)* is a directed graph where each node of the graph can be a basic atomic data type such as string or date, or a complex object type such as student. A root node represents a complex object type and is an entry point to the graph. The rest of the nodes in the graph are known as *component nodes*. A node that represents a basic atomic data type is also known as a *leaf node*. The atomic data type is attached as a label to the corresponding leaf node in the S3-Graph. Each directed edge in the S3-graph is associated with a *tag* which represents the relationship between the source node and the destination node. There are three types of edges. A node V_1 is connected to node V_2 by a *component edge* (denoted by a solid arrow line) with a tag T if V_2 is a component of V_1. If T is suffixed with a "*", the relationship is interpreted as "The entity type represented by V_1 has many T". Otherwise, the relationship is interpreted as "The entity type represented by V_1 has at most one T". A node V_1 is connected to another node V_2 via a *referencing edge* (denoted by a dashed arrow line) if V_1 references the entity represented by node V_2. A node V_1 is pointed by a *root edge* (denoted by a solid arrow line without any source node)

```
<Student>
    <Name>John</Name>
    <Course CRef="c1">
        <Grade>89</Grade>
        <Tutor>
            <TName>Tan</TName>
            <Office>S1605-13</Office>
            <Feedback>8</Feedback>
        </Tutor>
        <Tutor>
            <TName>Lim</TName>
            <Office>
                <Building>S17</Building>
                <Room>03-18</Room>
            </Office>
            <Office>S1504-26</Office>
            <Feedback>9</Feedback>
        </Tutor>
    </Course>
    <Course CRef="c2">
        <Grade>A</Grade>
        <Tutor>
            <TName>Tan</TName>
            <Office>S1605-13</Office>
            <Feedback>10</Feedback>
        </Tutor>
    </Course>
</Student>

<Course ID="c1">
    <Code>CS101</Code>
    <Title>Java</Title>
</Course>

<Course ID="c2">
    <Code>IT321</Code>
    <Title>Database</Title>
</Course>
```

Figure 1: Example of an XML document

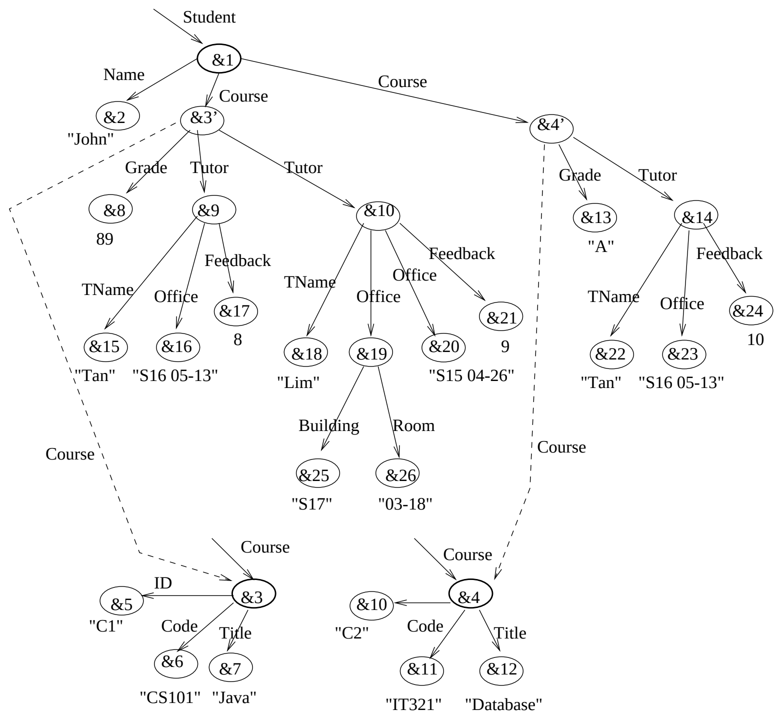

Figure 2: Semi-structured data graph of the XML document in Figure 1

with a tag T if the entity type represented by V_1 is owned by the database. V_1 is also known as a *root node* in the S3-Graph.

Figure 3 shows the schema of the semi-structured data graph in Figure 2. We differentiate nodes in a semi-structured data graph from the schema graph by using the & symbol instead of #. Node #1 is a root node representing the entity Student. This node is associated with the role "Student". Node #2 is both a component node as well as a leaf node. Node #3' is a component node which references the root node representing Course.

A *semi-structured data graph* D with respect to a S3-Graph G is a graph showing a database instance such that each node v in D is associated with one and only one node V of role R in G. V is called the *definition node* of the data node v. A component edge in D is represented by a solid arrow line. If v_1 is connected to v_2 via a component edge e with tag T in D, then their definition nodes in G must be likewise connected via a component edge E with tag T. E in G is called the *definition edge* of the edge e in D. Likewise for the referencing edge which is represented by a dashed arrow line. We say that the semi-structured data graph D is a data instance of G.

Figure 2 is a semi-structured data graph which is a data instance of the S3-Graph in Figure 3. The design is not a good one because there is data redundancy. Each tutor has one name, but

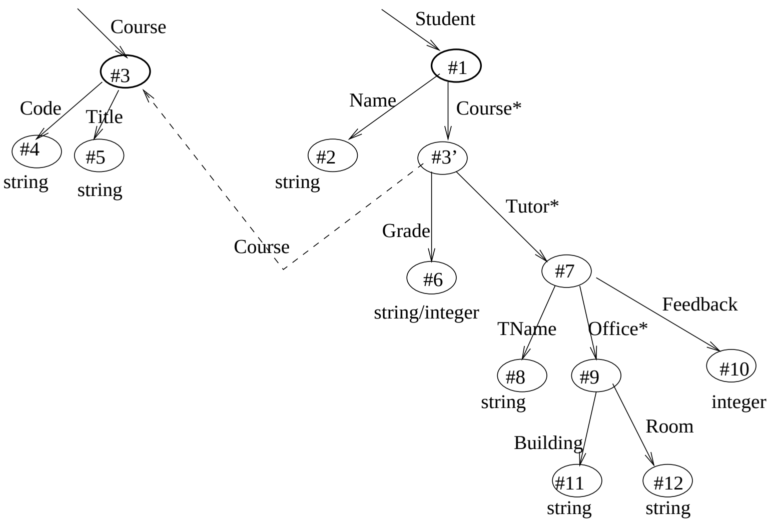

Figure 3: Schema of the Semi-Structured Graph in Figure 2

this information may appear many times in a semi-structured data graph as each tutor may teach many students. For example, the name "Tan" is repeated twice in the semi-structured data graph in Figure 2. The two instances "$\ll$ &9 $\gg$" and "$\ll$ &14 $\gg$" actually represent the same tutor "Tan" and his office "S16 05-13". The offices that a tutor occupies is solely dependent on the tutor, and is not dependent on the courses he/she teaches, nor the students he/she has. As a result, many anomalies can occur. If a tutor changes his/her office, then we must make sure that all occurences of this information are consistent. In order to remove the anomalies that may exist in semi-structured data, [10] employs a decomposition method to restructure a S3-graph. However, as in the case of relational database, the decomposition approach by no means ensures a good solution. Integrity constraint information can be lost during the decomposition.

2.3 The Entity-Relationship Approach

The ER approach [6] has been widely used in systems modeling, database design and even database integration [5, 2]. The ER model incorporates the concepts of entity types, relationship sets and attributes which correspond to structures naturally occuring in information systems and database applications. An *entity* is an object in the real world and can be distinctly identified. An *entity type* is a collection of similar entities which have the same set of predefined common properties or attributes. *Attributes* can be single-valued, multivalued or composite. Composite attributes are useful when we sometimes refer to the attribute as a unit, but other times refer specifically to its components. A minimal set of attributes whose values uniquely identify an entity in the entity type is called a *key.* There may be more than one key and one of them is designated as the primary key or *identifer.*

A *relationship* is an association among two or more entities. A *relationship set* is a collection of similar relationships which satisfy a set of predefined common attributes. The ER approach supports *recursive* relationships which involves entities in the same entity type. If the existence of an entity in one entity type depends on the existence of a specific entity in another entity type, such a relationship set and entity type are called *existence dependent relationship set* and *weak entity type.* A special case of existence dependent relationship set occurs if the entities in an entity type cannot be identified by the values of their own attributes, but has to be identified by their relationship with other entities. Such a relationship set is called an *identifier dependent relationship set.* A relationship set which involves weak entity type(s) is called a weak relationship set. An entity type which is not weak is called a regular entity type. If an entity in one entity type $E1$ is also in another entity type $E2$, we say that $E1$ ISA $E2$. Relationship sets can be viewed as a high level entity type known as *aggregation.*

[11] defined a *normal form* ER diagram which captures and preserves the dependencies of the real world, in terms of functional, multi-valued and join dependencies, by representing them explicitly in the ER diagram. All the relationships represented in a normal form ER diagram are non-redundant, i.e. none of the relationships can be derived from other relationships. This ensures that the relations translated directly from the normal form ER diagram are in good normal form, either in 3NF or 5NF. [11] also give an algorithm to transform ER diagrams into their normal form.

Figure 4 shows the ER diagram for our Student-Course-Tutor example. We see that students, courses and tutors are modeled as entity types Student, Course and Tutor respectively. Student has an attribute Name while Course has attributes Code and Title. Tutor has a single-valued attribute Name and a composite multivalued attribute Office. Here, we assume that Student.Name, Course.Code and Tutor.Name are the identifiers of the entity types Student, Course and Tutor respectively. The relationship set Enrol captures the association that a student is enrolled in a course and has a single-valued attribute Grade. Since a student taking a course is taught by some tutors, we need to associate the relationship set Enrol with the entity type Tutor. This is accomplished using aggregation which effectively allows us to view Enrol as an entity type for the purpose of participation in other relationship sets. This association is captured in the relationship set SCT. Feedback is a single-valued attribute in SCT as its value is given by the student for each tutor teaching him in some course. The ER diagram is also in normal form.

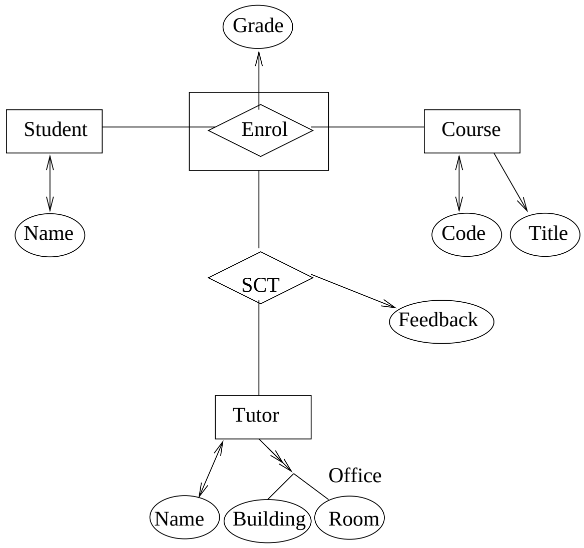

Figure 4: Entity-Relationship Diagram for Student-Course-Tutor Example

3 ER to S3-Graph Translation

The task of designing a "good" semi-structured database can be made easier if we have more semantics. We have seen how the ER model allows us to describe data at a high level of abstraction. [11] proposes a normal form for the ER model and gives a method to transform ER diagrams to their normal form. [12] uses the ER normal form to design normal form nested relations. A set of nested relations is in normal form (NF-NR) if we translate this set of nested relations into relations which are at least in third normal form. This top-down approach [12], which consists of normalizing an ER diagram and converting a normalized ER diagram into a set of normal form nested relations, has two advantages. First, normalizing an ER diagram effectively removes ambiguities, anomalies and redundancies at a semantic level. Second, converting a normalized ER diagram into a set of nested relations results in a database schema with clean semantics, in a good normal form.

We observe that an S3-graph is similar to the nested relational and hierarchical model in that they have a tree-like structure and allow repeating groups or multiple occurences of objects. While all these models represent hierarchical organizations in a direct and natural way, they are problematic when representing situations with nonhierarchical relationships. Duplication of data is necessary when we try to represent many-to-many relationships or relationships that involve more than two participating entity types. The problem of handling symmetric queries with data duplication also exists in semi-structured data. Keys and foreign keys are used in nested relations while virtual pairing by logical parent pointers to represent many-to-many relationships are used in the hierarchical model to allow symmetric queries. In S3-graphs, we will show how referencing edges can be used to remove redundancies.

The rest of this section will describe the translation of an ER diagram to an S3-Graph. If the ER diagram is not in normal form, then the S3-Graph obtained may contain redundancy. However, if the ER diagram is in normal form, then the S3-Graph obtained will also be in normal form (S3-NF). This effectively removes anomalies in the resulting semi-structured database and any redundancy due to many-to-many relationships and n-ary relationships will be controlled.

Algorithm: Translation of an ER diagram to a S3-Graph.
Input: an ER diagram
Output: the equivalent S3-Graph
Step 1. Transform the ER diagram to a normal form ER diagram.
Step 2. Map regular entity types.
Step 3. Map weak entity types.
Step 4. Map relationship sets which do not have aggregations as participating entity types.
Step 5. Map relationship sets which have aggregations as participating entity types.
Step 6. Map ISA relationship sets.

We will elaborate on each of these steps.

Step 1. *Transform the ER diagram to a normal form ER diagram.*

Detailed steps with examples are described in [11].

Step 2. *Map regular entity types.*

Each regular entity type E becomes the root node N of an S3-Graph. N is pointed to be a root edge with a tag E.

(a) Each single-valued and multivalued attribute A of E is mapped to a component node N_A which is connected to E by a component edge with tag A. If A is a multivalued attribute, then the tag is suffixed by an "*". N_A is also a leaf node whose label is the data type of attribute A.

(b) Each composite attribute A of E is mapped to a component node N_A which is connected to E by a component edge with tag A. If A is multivalued, then the tag is suffixed by an "*". Each component attribute C of A is mapped to a component node N_C which is connected to N_A by a component edge with tag C. N_C is also a leaf node whose label is the data type of component attribute C.

Step 3. *Map weak entity types.*

Each weak entity type W becomes a component node N_W which is connected to the root node N corresponding to its owner entity type E by a component edge with tag W suffixed by an "*". The attributes of W are mapped to component nodes which are connected to N_W by component edges in the same way as the attributes of a regular entity type.

Step 4. *Map regular relationship sets which do not have aggregations as participating entity types.*

The participating entity types of a relationship set R may be aggregations or simply entity types. Here, we first consider relationship sets which do not have aggregations as participating entity types.

Case (1) *R is a binary relationship set.*

Let R be a binary relationship set with participating entity types E_A and E_B. Let E_A and E_B be mapped to root nodes N_A and N_B respectively. Depending on the application, there are several ways to map R:

(a) R is mapped to a component node N_R.
N_A is connected to N_R by a component edge tagged R_A and N_R connected to N_B by a referencing edge tagged R_B.

(b) R is mapped to a component node N_R.
N_B is connected to N_R by a component edge tagged R_B and N_R connected to N_A by a referencing edge tagged R_A.

(c) R is mapped to component nodes N_R and $N_{R'}$.
N_A is connected to N_R by a component edge tagged R_A and N_R connected to N_B by a referencing edge tagged R_B, while N_B is connected to $N_{R'}$ by a component edge tagged $R'_B$and $N_{R'}$ connected to N_A by a referencing edge tagged R'_A.

(d) R is mapped to a root node N_R.
N_R is connected to N_A and N_B by referencing edges tagged R_A and R_B respectively.

If R is a one-to-one relationship set, then all the tags of the component and referencing edges are not suffixed by an "*". If R is a many-to-many relationship set, then the tags of the component edges are suffixed by an "*". If R is a one-to-many relationship set, then without loss of generality, let the cardinalities of E_A and E_B in R be 1 and m respectively. If we have N_A connected to N_R by a component edge, then the tag of the component edge is suffixed by an "*". Otherwise, if we have N_B connected to N_R by a component edge, then the tag of the component edge is suffixed by an "*". The tags of the component edges for $N_{R'}$ are similarly suffixed.

Any attributes of R are mapped to component nodes connected to N_R by component edges in the same way as the attributes of regular entity types. If R is mapped using (c), then the attributes are duplicated and mapped to component nodes connected to $N_{R'}$ by component edges. Figure 5 summarizes the mapping of binary relationship sets. We note that mappings (a) and (b) do not allow symmetric queries to be answered efficiently, while mapping (c) has controlled data redundancy. The relationship set R is "flattened" in mapping (d) to allow symmetric queries without duplication.

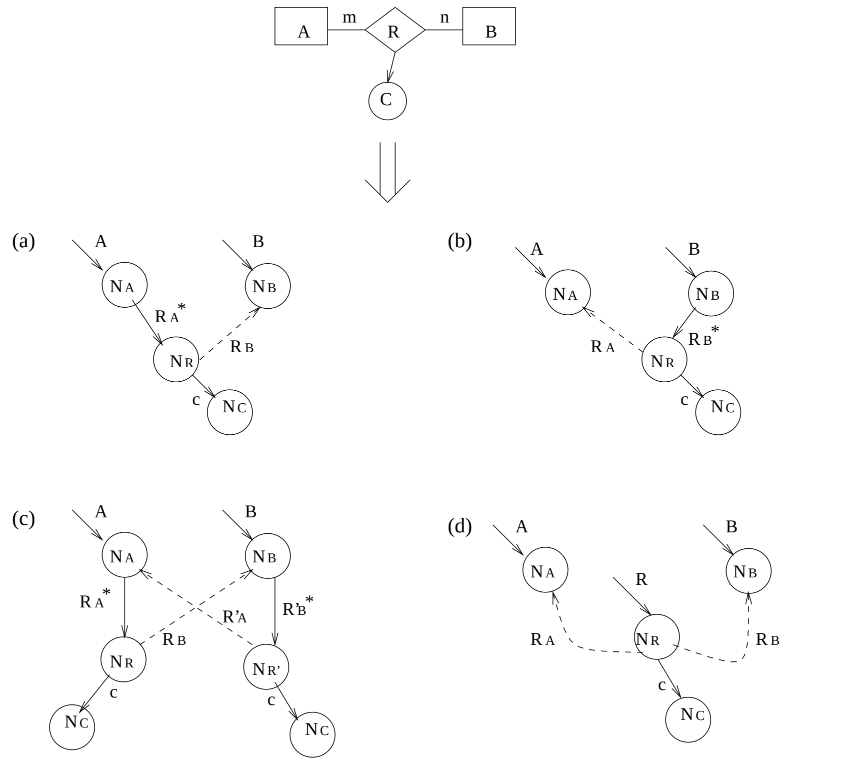

Figure 5: Different ways to map a binary relationship set in an ER diagram to an S3-graph

Case (2) *R is a recursive relationship set.*

If R is a recursive relationship set, then only one entity type, say E_A is involved in R with roles r1 and r2. Again, depending on the application, there are several ways to map R:

(a) R is mapped to a component node N_R.

N_A is connected to N_R by a component edge tagged $r1$ and N_R connected to N_A by a referencing edge tagged $r2$.

(b) R is mapped to a component node N_R.

N_A is connected to N_R by a component edge tagged $r2$ and N_R connected to N_A by a referencing edge tagged $r1$.

(c) R is mapped to component nodes N_R and $N_{R'}$.

N_A is connected to N_R by a component edge tagged $r1$ and N_R connected to N_A by a referencing edge tagged $r2$, while N_A is connected to $N_{R'}$ by a component edge tagged $r2$ and $N_{R'}$ connected to N_A by a referencing edge tagged $r1$.

(d) R is mapped to a root node N_R.

N_R is connected to N_A by 2 referencing edges tagged $r1$ and $r2$ respectively.

The tags of the component edges may be suffixed with a "*" in the same way as for the binary relationship sets. Any attributes of R are mapped to component nodes connected to N_R by component edges in the same manner as the attributes of binary relationship sets. Figure 6 summarizes the mapping of recursive relationship sets. Again, we note that mappings (a) and (b) do not allow symmetric queries to be answered efficiently, while mapping (c) has controlled data redundancy. The relationship set R is "flattened" in mapping (d) to allow symmetric queries without duplication.

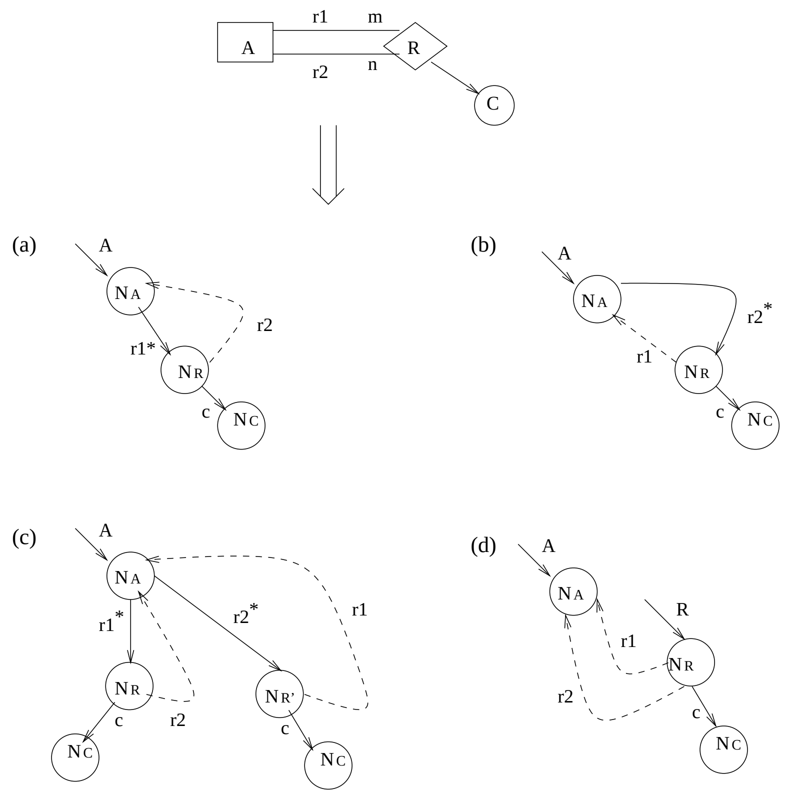

Figure 6: Different ways to map a recursive relationship set in an ER diagram to an S3-graph

Case (3) *R is a n-ary relationship set where $n > 2$.*

Let R be an n-ary relationship set with participating entity types E_1, E_2, ..., E_m, where $m > 2$. Let E_1, E_2, ..., E_m be mapped to root nodes N_1, N_2, ..., N_m respectively. There are several ways to map R depending on the application:

(a) R is mapped to a component node N_R. Without loss of generality, we connect N_1 to N_R by a component edge tagged R_{E1}. Then N_R has a referencing edge tagged $R_{E1}, R_{E2}, ..., R_{Em}$ to each of the root nodes N_2, N_3, ..., N_m respectively. The tag of the component edge is simply R_{E1} if the cardinality of all the participating entity types in R is 1, otherwise it is suffixed by an "*". Each of the referencing edge to N_i, where $2 \leq i \leq m$, is simply tagged R_{Ei}. Any attributes of R are mapped to component nodes connected to N_R by component edges in the same way as the attributes of regular entity types.

(b) An alternative approach to map an n-ary relationship set R is to first choose a path to link the participating entity types of R. Let $\prec V_1, V_2, V_3, \cdots, V_k \succ$ be the path, where vertex V_1 corresponds to some participating entity type of R which is associated with some root node N_1, and vertex V_i, $2 \leq i \leq k$, corresponds to either a participating entity type of R or a combination of two or more participating entity types of R. Next, we create component nodes $N_{R2}, N_{R3}, ..., N_{Rk}$ that is associated with $V_2, V_3, ..., V_k$ respectively. Then, root node N_1 has a component edge tagged ${R_2}_{E1}$ to node N_{R_2}, while each node N_{R_i}, where $2 \leq i \leq k-1$, has a component edge tagged ${R_{i+1}}_{E1}$ to $N_{R_{i+1}}$, and a referencing edge(s) tagged R_{Ei} to the root node(s) that is associated with the participating entity type(s) of R corresponding to V_i. The tags of the component edges are suffixed by an "*" if the cardinalities of all the participating entity types in R are not all 1. Any attributes of R are mapped to component nodes connected to N_{R_k} by component edges in the same way as the attributes of regular entity types.

(c) "Flatten" the relationship set by mapping R to a root node N_R. N_R is connected to N_1, N_2, ..., N_m by referencing edges tagged $R_{E1}, R_{E2}, ..., R_{Em}$ respectively. Any attributes of R are mapped to component nodes connected to N_R by component edges in the same way as the attributes of regular entity types.

Figure 7 summarizes the different ways we can map an n-ary relationship set in an ER diagram to an S3-graph.

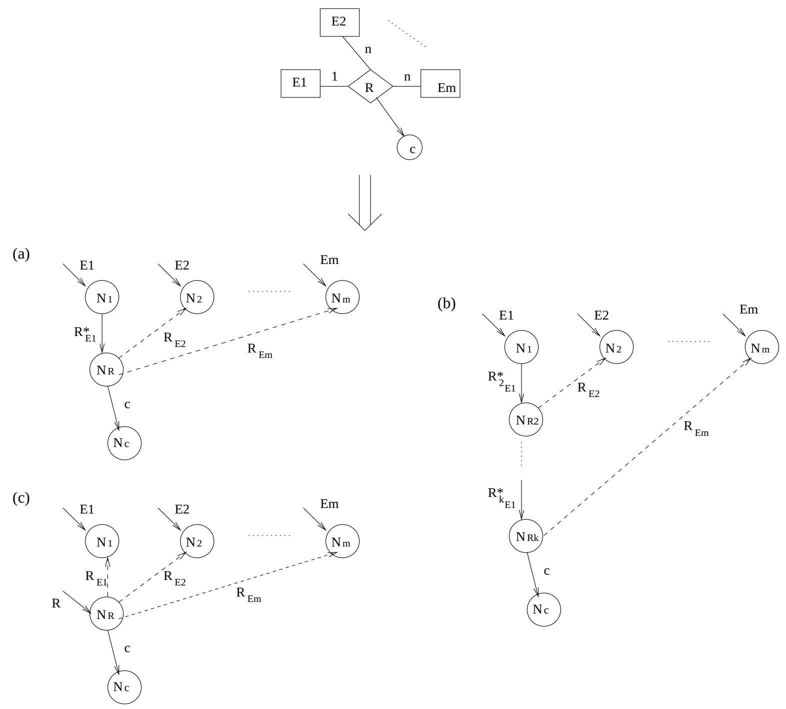

Figure 7: Different ways to map an n-ary relationship set in an ER diagram to an S3-graph

Step 5. *Map regular relationship sets which have aggregations as participating entity types.*

Aggregations are a means of enforcing inclusion dependencies in a database. There are two ways to map a relationship set involving aggregations depending on whether it is necessary for an application to enforce the constraint.

Let R be the regular relationship set. Let E_1, E_2, ..., E_m and E_{A_1}, E_{A_2}, ..., E_{A_n} be the participating entity types of R, where E_1, E_2, ..., E_m have been mapped to root nodes N_1, N_2, ..., N_m respectively, and E_{A_1}, E_{A_2}, ..., E_{A_n} are also aggregations.

(a) Enforce the inclusion dependencies due to aggregations.

Each E_{A_i}, where $1 \leq i \leq n$, is mapped to a root node N_{A_i} using Step 3 above. R is mapped to a root node N_R. N_R is connected to N_1, N_2, ..., N_m, $N_{A_1}, N_{A_2}, ..., N_{A_n}$ by referencing edges tagged $R_{E1}, R_{E2}, ..., R_{Em}, R_{A1}, R_{A2}, ..., R_{An}$ respectively.

(b) Do not enforce the inclusion dependencies due to aggregations.

Each E_{A_i}, where $1 \leq i \leq n$, is mapped to a node N_{A_i} using Step 3 above, where N_{A_i} can be a root node or a component node. R is mapped to a root node N_R. N_R is connected by referencing edges to N_1, N_2, ..., N_m, and the root nodes corresponding to the participating entity types of the aggregations E_{A_1}, E_{A_2}, ..., E_{A_n}.

Any attributes of R are mapped to component nodes connected to N_R by component edges in the same way as the attributes of regular entity types. Figure 8 illustrates how a relationship set which has aggregations as its participating entity types can be translated.

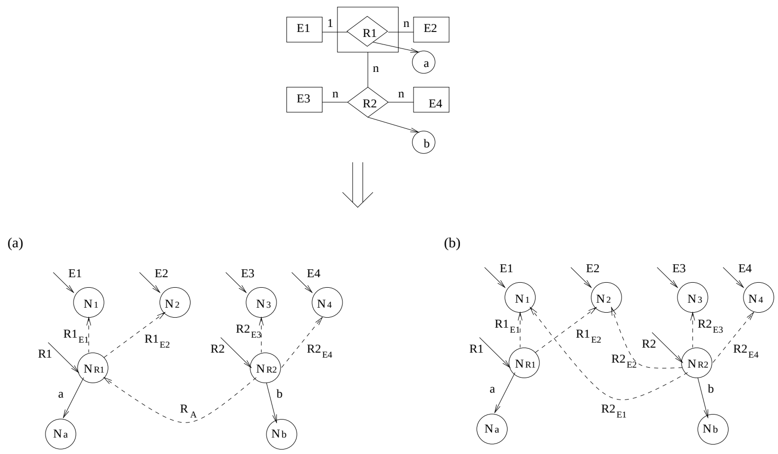

Figure 8: Different ways to map relationship sets involving aggregations in an ER diagram to an S3-graph

Step 6. *Map special relationship set ISA.*

Given two entity types A and B where A ISA B. We map A and B to entity nodes N_A and N_B respectively. The ISA relationship set is mapped to a referencing edge that connects N_A to N_B. Note that the referencing edge is tagged ISA which has a special meaning. Figure 9 shows the translation of ISA relationship sets.

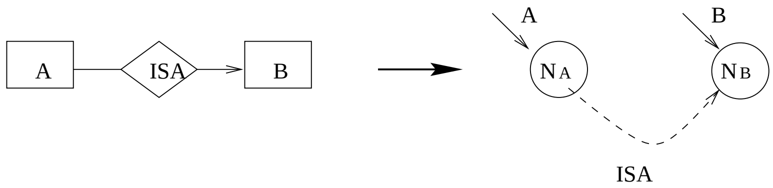

Figure 9: Mapping an ISA relationship set in an ER diagram to an S3-graph

Example 3.1 The ER diagram in Figure 4 can be translated to the semi-structured schema graph in Figure 10 as follows. The entity types Student, Course and Tutor become entity nodes #1, #3, #7 respectively. The attributes also become nodes and are connected to their owner entity type by component edges. We need to process the relationship Enrol before SCT because Enrol is involved in an aggregation. Enrol is mapped to an entity node #13 with component edges to entity nodes Course and Student. The attribute Grade is a component of the entity node #13. Next, we map the relationship set SCT to an entity node #15 which has component edges to entity nodes Enrol and Tutor. The attribute Feedback is a component of node #15. The S3-Graph obtained does not contain data redundancy. Note that the relationship sets Enrol and SCT have been flattened in the S3-Graph in Figure 10 because we want to answer symmetric queries with no redundant data. However, if an application only process queries which retrieve the courses taken by a student, and do not need to find students who take a given course, then Figure 11 shows an alternative way to map the ER diagram. Note that this schema cannot answer symmetric queries effectively. □

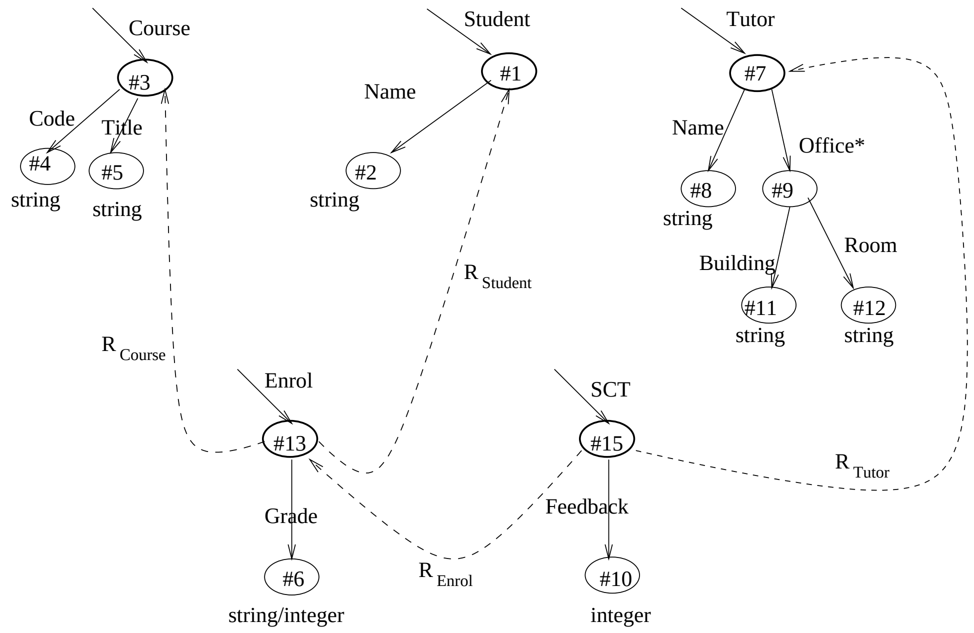

Figure 10: An S3-Graph for the ER diagram in Figure 4

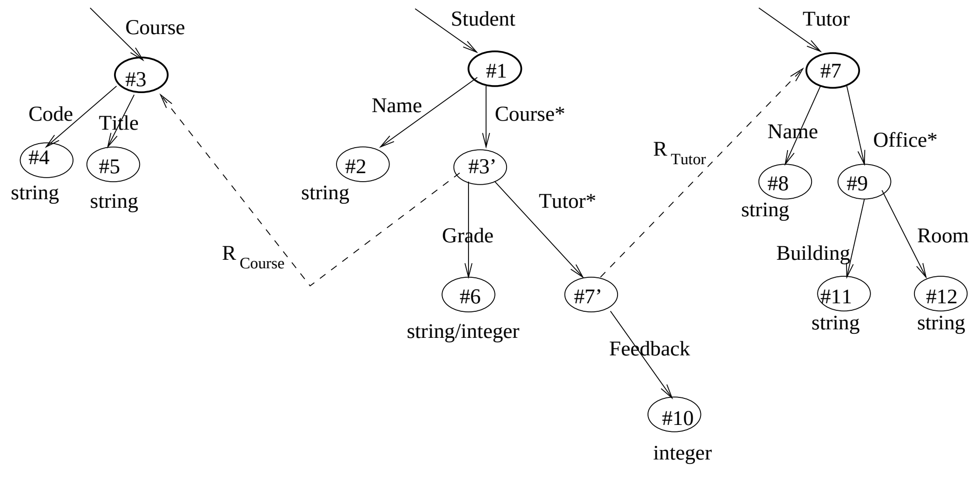

Figure 11: An alternative S3-Graph for the ER diagram in Figure 4

4 Conclusion

To the best of our knowledge, this is the first paper that presents a conceptual approach for designing semi-structured databases which can be associated with some schema. We envisage the growing importance of well designed semi-structured databases with the development of new e-commerce applications that require the efficient design and maintenance of large amounts of data. The introduction of an ER-based conceptual layer allows us to remove anomalies and data redundancies at the semantic level. We have developed an algorithm to map an ER diagram involving weak entity types, recursive, n-ary and ISA relationship sets, and aggregations to a normal form S3-Graph. Using the mappings proposed, XML DTDs and customised XML views can be generated from the normal form ER diagrams. In addition, relational tables can be created from the normalized ER diagram to store the XML data with controlled or no redundancy. We have also compared the S3-Graph to the hierarchical model and nested relations. Given their tree-like structures, it is not easy for these models to represent nonhierarchical relationships to support symmetric queries without introducing data redundancies. Our approach allows an application to have various options to represent n-ary and many-to-many relationship sets in an S3-Graph.

References

[1] S. Abiteboul, D. Quass, J. Widom, and J. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1), 1997.

[2] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity-relationship model. *IEEE Trans. on Software Engineering*, 10:650–664, 1984.

[3] T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible markup language (xml) 1.0. W3C Recommendation available at http://www.w3.org/TR/1998, 1998.

[4] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization technique for unstructured data. In *Proc. ACM SIGMOD*, 1996.

[5] E.P.F. Chan and F.H. Lochovsky. A graphical data base design using the entity-relationship model. In *Entity-Relationship Approach to Systems Analysis and Design, North Holland, Chen, P.P. (ed)*, pages 295–310, 1980.

[6] P.P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):166–192, 1976.

[7] Document Object Model (DOM) Level 1 Specification. *http://www.w3.org/TR/REC-DOM-Level-1.*

[8] R. Elmasri and S.B. Navathe. Fundamentals of database systems. Benjamin and Cummings Publishing Company, 1989.

[9] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3), 1997.

[10] S.Y. Lee, M.L. Lee, T.W. Ling, and L. Kalinichenko. Designing good semi-structured databases. In *Proc. of 18th Int. Conference on Entity-Relationship Approach*, 1999.

[11] T.W. Ling. A normal form for entity-relationship diagrams. In *Proc. of 4th Int. Conference on Entity-Relationship Approach*, pages 24–35, 1985.

[12] T.W. Ling. A normal form for sets of not-necessarily normalized relations. In *Proc. of 22nd Hawaii Int. Conference on Systems Science*, pages 578–586, 1989.

[13] J. McHugh, S. Abiteboul, R. Goldman, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3), 1997.

[14] G. Mecca, P. Merialdo and P. Atzeni. Araneus in the Era of XML. *Bulletin of IEEE Computer Society Technical Committee on Data Engineering*, 1999.

[15] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *IEEE International Conference on Data Engineering*, pages 251–260, 1995.

[16] T. Teorey. Database Modeling and Design: The ER Approach. Morgan Kaufmann, 1990.