European Journal of Scientific Research
ISSN 1450-216X Vol.44 No.2 (2010), pp.314-336

http://www.eurojournals.com/ejsr.htm

XML Document Design via GN-DTD

Zurinahni Zainol
Department of Computer Science, University of Hull, HU6 7RX, UK
School of Computer Science, Universiti Sains Malaysia, 11800, Malaysia
E-mail: zuri@cs.usm.my; or z.zainol@2007.hull.ac.uk

Bing Wang
Department of Computer Science, University of Hull, HU6 7RX, UK
E-mail: b.wang@hull.ac.uk

Abstract

Designing a well-structured XML document is important for the sake of readability and maintainability. More importantly, this will avoid data redundancies and update anomalies when maintaining a large quantity of XML based documents. In this paper, we propose a method to improve XML structural design by adopting graphical notations for Document Type Definitions (GN-DTD), which is used to describe the structure of an XML document at the schema level. Multiples levels of normal forms for GN-DTD are proposed on the basis of conceptual model approaches and theories of normalization. The normalization rules are applied to transform a poorly designed XML document into a well-designed based on normalized GN-DTD, which is illustrated through examples.

Keywords: XML model and design, graphical notations, DTD, XML Normal Forms

1. Introduction

With the wide exploitation of the web and the accessibility of a huge amount of electronic data, XML (extensible Markup Language) has been used as a standard means of information representation and exchange over the web. Additionally, XML is currently used for many different types of applications which can be classified into two main categories [21, 22]. The first application is called *document centric* XML and the other is called *data centric* XML. The document centric XML is used as a markup language for semi-structured text documents with mixed-content elements and comments. Furthermore, the order of sibling elements is significant, for instance, in user's manual, webpage, etc. The data centric XML consists of regular structure data for automated processing and there are little or no element with mixed content, comments, and processing instruction for instance, geography, e-commerce, and scientific data [8, 23]. In this work, we focus on data centric applications.

In data centric applications, data redundancy and update anomalies are the most significant problems. As stated by Kolahi [11], when data centric document applications are presented in XML, it is unavoidable that data redundancy and update anomalies will appear. Data redundancy and update anomalies occur in XML documents if their type structures, such as DTDs, are not well-formed. These problems are similar to those in relational databases [1, 6, 23]. To better understand this problem in XML applications specifically, let us consider a university database which describes departments, courses, students, lecturers and their relationships as shown in the following figures (Figure 1 and Figure 2).

The DTD shown in Figure 1 has the following information:

- Each department offers many courses indicated by the notation *.
- Every course is described by the attribute course no (cno), title and numbers of students taking the course.
- Each student has a student number (sno), first name or last name as optional and an assigned lecturer.
- Each lecturer has his/her number (tno), and name (tname).

Figure 1: DTD

```
<!DOCTYPE department[
    <!ELEMENT department(course*)>
<!ELEMENT course(title, student*)>
      <!ATTLIST course cno ID #REQUIRED>
   <!ELEMENT title (#PCDATA)>
   <!ELEMENT student(fname|lname?,lecturer)>
      <!ATTLIST student Sno ID #REQUIRED
   <!ELEMENT fname(#PCDATA) >
   <!ELEMENT lname(#PCDATA) >
   <!ELEMENT lecturer (tname)>
      <!ATTLIST lecturer tno ID #REQUIRED>
   <!ELEMENT tname (#PCDATA)>
]>
```

Figure 2: XML document conforms to DTD in Figure 1

```
<!DOCTYPE Department [
 <course>
    <course cno = "csc101">
    < title > XML database </title>
    < student >
        <student sno = "112344">
        <fname> David</fname>
        <lname> Grey </lname>
        <lecturer>
            <lecturer tno = "123">
            <tname>Bing </tname>
         </lecturer>
     </student>
    < student >
        <student sno = "112345">
        <fname>Helen</ftname>
         <lecturer>
            <lecturer tno = "123">
            <tname> Bing </tname>
        </lecturer>
     </student>
</course>
<course>
    <course cno = "csc102">
    < title > Z formal methods </title>
     < student >
        <student sno = "112344">
        <fname> David</fname>
        <lname>Grey </lname>
        <lecturer>
            <lecturer tno = "123">
         <tname> Bottaci  </tname>
         </lecturer>
     </student>
    < student >
        <student sno = "112345">
        <fname>Helen </fname>
         <lecturer>
            <lecturer tno = "123">
            <tname> Bottaci </tname>
        </lecturer>
     </student>
</course>
</Department>]
```

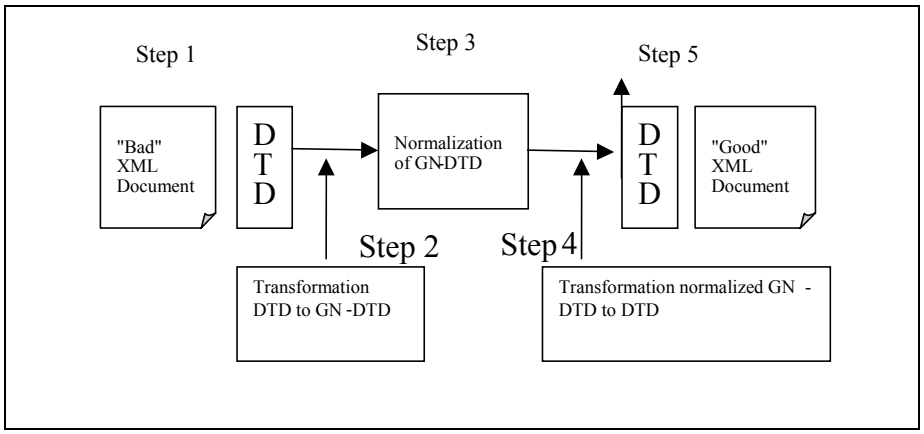
Any XML document that satisfies and conforms to this DTD is likely to contain data redundancies which may lead to update anomalies. For example, as shown in Figure 2, the lecturer named *Bing* who teaches the same *course number* (cno) *csc101* is stored twice, which will lead to the problems described above. To avoid such problems, a set of rules should be provided when designing a DTD for XML documents.

In relational database design, normalization rules are used to help designers to design a good relational database [5]. By having these rules in mind, designers can follow either of two methodologies to design a relational database: the conceptual approach [4] or the normalization theory approach [5]. These similar approaches with some modification have been applied to XML schema design by XML database researchers. However, the nature of XML documents is different from relational database. The task of designing XML documents is becoming more challenging than designing relational databases since an XML document is hierarchical, irregular, and often associated with Document Type Definition (DTD) [1, 21].

The objective of this work is to provide a methodology which simplifies the process of designing a non redundancy XML document. To achieve this, a conceptual model called GN-DTD is proposed. GN-DTD is a graphical modelling approach for describing both DTD and XML documents. For GN-DTD itself, we define a complete set of syntax and structure which incorporates attributes, simple data type, complex data type, and types of relationship among them. Furthermore, semantic constraints are also precisely defined in order to capture semantic meaning among those defined objects.

In this work, we present normal forms for GN-DTD based on both Arenas and Libkin's rules [1] and Ling et al's rules [15] in a simpler form to allow users/designers to find an 'optimal' structure of XML elements/attributes. This will produce a correct, complete and consistent representation of the real world XML data which may benefit the users. We ensure that DTD mapped from GN-DTD are similar to XNF [1]. Finally, we propose normalization and mapping rules to transform from normalised GN-DTD back to its new DTD.

Figure 3: XML document design process



The proposed design process of XML documents via GN-DTD is illustrated in Figure 3. This process, roughly, takes DTD as input and converts it to GN-DTD. Normalization is carried out automatically based on the number of data dependencies provided by the user at the conceptual model. Finally, the XML document is generated on the basis of normalized GN-DTD.

The rest of the paper is organized as follows. Section 2 provides information on related works of conceptual models and normalization theory. Section 3 discusses the notation of GN-DTD. Section 4 presents a normal form of GN-DTD. Section 5 provides illustrative examples to convert from an un-normalized GN-DTD to a normalized one using a normalization algorithm. Section 6 shows how GN-DTD can also be transformed back to a new DTD structure. We conclude the paper with conclusions and suggestion for future work in Section 7.

2. Related Work

Two main approaches have been applied by XML database researchers to design non- redundant XML documents which is a conceptual data modeling approach [7, 12, 15, 17, 24] and normalization theory [1, 11,16, 21, 22, 23]. Both of these approaches are based on relational database design theory [4, 5]. Conceptual approach involves a two-phase process: *conceptualization*, in which XML data is first displayed in terms of a conceptual model suitable for the of domain experts to understand, then the model is restructured to eliminate redundancy by using normalization rules and finally mapping the model into an XML schema. On the other hand, the normalization theory approach ignores conceptualization, instead of representing the XML document directly using DTD with a set of data dependencies. Generally data dependency such as functional dependency is used to detect data redundancy in the XML document during the schema design process [1, 11, 22, 23]. Due to this, many notions XML functional dependencies have been proposed to be used in schema normalization [1, 12, 22, 23]

2.1. Conceptual Data Modeling Approach

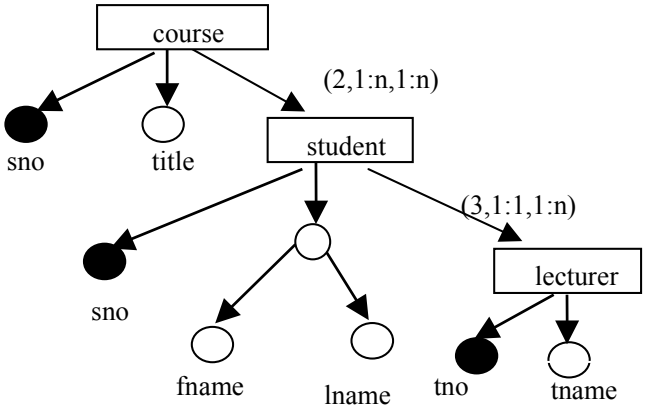
Many contributions have been made to develop a conceptual model for XML documents. As XML has a close similarity to semi structured models [3, 6], the trend in the literature is to apply data models developed for semi structured to XML. Most XML data models currently use directed edge labeled graphs to represent XML documents and their schemas. These models consist of nodes and directed edges which respectively represent XML elements in a document and relationships among the elements. In this work, we focus on the data models defined for data management which capture the constraints explicitly. Data models such as S3-graph [13], CM Hypergraph [7], ORA-SS[6] and Semantic Network [8] have been used for XML design. In this work, we focus on ORA-SS as it is most related to our work.

Dobbie et al [6] defined a semi-structured data model, Object Relational Attribute-Semi-Structured (ORA-SS) to represent data conceptually. ORA-SS distinguishes between object, relationships and attributes and reference object. Object is like the entity type in a traditional ER diagram [4] and a relationship between object is expressed through a directed arrow labelled with a degree of n-ary relationship (n>2). Objects are represented as labelled rectangles, attributes as labelled circles and primary key attributes are denoted as filled circles. Attributes represent the properties of the object while a reference is denoted by a dashed edge between a reference object. The ORA-SS notation is similar to the ER model but with a hierarchical representation. The advantage of this approach is that the major attribute dependencies between attribute keys (unique identifier) are captured in the modelling process.

Ling et al [15] proposed a methodology to design semi structured databases using ORA-SS schema diagrams. Given semi structured data, like an XML document, the ORA-SS schema is extracted from the XML document using extraction rules [15]. Then, the ORA-SS schema is converted into normal form ORA-SS. Figure 4 illustrates an ORA-SS schema extracted from XML document in Figure 2 using extraction rules [15]. However, in the ORA-SS schema diagram, root element in the XML document is not extracted as an object class. ORA-SS shown in Figure 4 is not in a normal form because it violates some of the features of ORA-SS normal form.

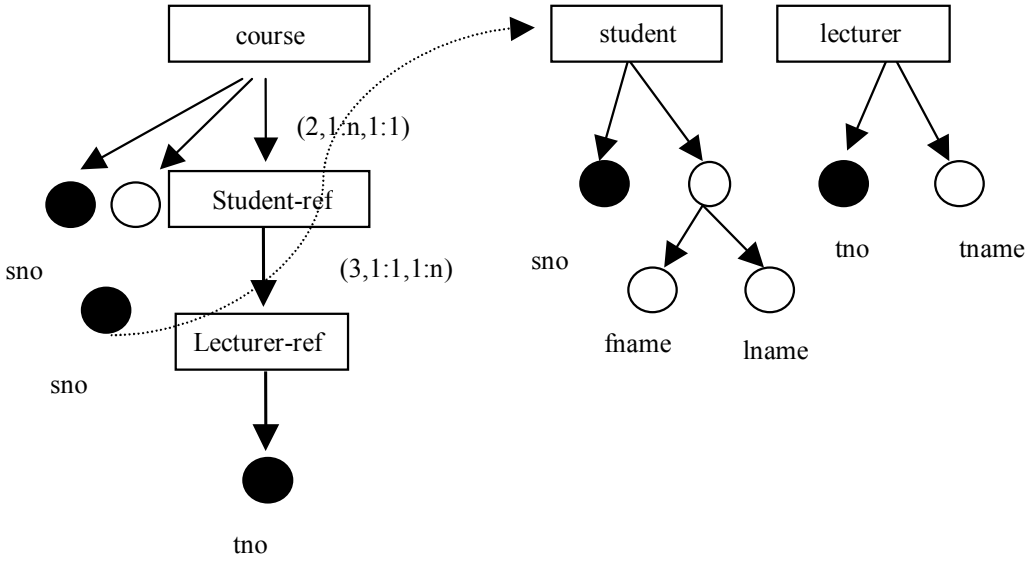
The concept of normal form ORA-SS depends on the twin concepts of an object class normal form (O-NF) and a relationship type normal form (R-NF) which is an extension to the NF-NR for nested relation [13]. The ORS-SS normal form reduces data redundancies using semantic relationship between the objects based on 3NF and 4NF. This approach differs from Arenas and Libkin's approach, because they take constraints from the conceptual model rather than from specified XML functional dependency (XFD). The nature of the definition for the normal form ORA-SS depends on a number of conditions. First, none of the attributes of the object class have multi-value or transitive dependency on the key of object class and relationship type. Second, every nested object class and relationship within the parent object class must non redundant.

Figure 4: ORA-SS schema diagram



Ling et al [15] have proposed an algorithm to convert the ORA-SS schema diagram to a normal form ORA-SS schema diagram. Using their normalization algorithm [15] a normal form ORA-SS as shown in Figure 5 is derived.

Figure 5: A normal form ORA-SS



Another data model similar to ORA-SS is Semi-Structured Schema Graph (S3-Graph) defined by Lee et al. [12]. Unlike ORA-SS, S3-Graph is unable to model the semantic relationship needed for recognizing redundancies in a semi-structured database. Lee et al. [12] also defined a normal form for S3-Graph, S3-NF. Embley and Mok [7] proposed a normal form for an XML document called XNF (XML normal form) based on the conceptual schema CM-Hyper graph.

There also exists XML models that are based on the traditional conceptual models, for example, Extended Entity Relational (EER) models [17]. They extend the ER model to model the structure of XML data. On the other hand, Bird [2] used an ORM to model XML Schema. However nesting was not considered in their work. We observed that ORM is an attribute free model; thus it is not suitable to represent DTD structure as in our work attribute must be displayed explicitly. In addition to that, Conrad et al. anticipated a unified modeling language (UML) to model XML schema. The mapping between the static part of the UML specification and XML DTD was developed. The disadvantage of UML notation is, that it does not have the graphical notation necessary to indentify that the attribute is unique and inclusive-or constraint [10].

2.2. Normalization Theory

The normalization approach was proposed by Codd [5] identifie data dependencies such as functional dependency that cause data redundancy in a relational database. Functional dependency helps the relational database designer to understand the redundancies that are present in a design. In principle, normalization theory for XML has been proposed similar manner to relational normalization [1, 11, 22, 23]. Even though there are some fundamental differences between relational models and XML models, similar techniques have been used. Many normal forms have been proposed based on XML functional dependency (XFD) definition [1, 11, 22, 23] and multivalued dependency definition [21]. However, the most fundamental and accepted XML normal form is proposed by Arenas and Libkin [1]. They propose a formal notion of XML functional dependency (XFD) using tree tuple based on ideas from relational schema [5] and nested relational schema [18]. Moreover, Arenas presented a formal model for an XML document as an XML labelled tree and DTD. The XML tree and DTD are defined precisely in a textual representation. For instance, Arenas and Libkin[1] define DTD as follows:

DTD, $D = (E,A,P,R,r)$ where;

1. E is a set of element sets
2. A is a set of attributes using and illustrated as symbol @ and PCDATA as S
3. P is a mapping from a set of elements to the children of the element set, indicating the semantic constraint of children (* ,?, +)
4. R is a mapping from a set of elements to a set of attributes
5. r is a root element set.

To demonstrate their idea, consider the DTD in Figure 1. The set E contains all element sets such as department, courses, title, fname, lname, lecturer and tname. The set A contains all attribute sets such as @sno, @cno, @tno. P maps from the element set to the children of the element set such as P(department) is course*, P(course) is student*, P(student) is { fname, lname, lecturer}, and P(fname) is S. R maps each element set to its attribute such as R(course) is sno, R(student) is cno and R(lecturer) is tno. The root element r is mapped to department.

Many constraints are captured in functional dependencies definition based on this formal model XML tree and DTD. XML functional dependency (XFD) is defined by considering a relational representation of an XML document. For example, the XFD (constraint) that two lecturer elements with same lecturer number (tno) value must have the same name is expressed as follows:

$$department.course.student.lecturer@tno \rightarrow department.course.student.lecturer.tname.S \quad (1)$$

where, *department.course.student.lecturer.tno* is a Left Hand Side(LHS) paths and

department.course.student.lecturer.tname.S is a Right Hand Side(RHS) path

Arenas & Libkin [1] defined that DTD is in a normal form (XNF) if every XML functional dependency defined over DTD is in XNF. The XFD is XNF if every (LHS) path can determine a

unique value of (RHS) path. This means that for every attribute value given; only one value of element set can be available in the XML document.

Based on this definition, XFD in *(1)* is not in XNF since the value for the element lecturer name is not unique for the attribute given, as lecturer name *Bing* appears twice in the XML document as shown in Figure 2. Because of this XFD, DTD is not in XNF. To eliminate this type of XFD, Arenas & Libkin [1] proposed an XNF decomposition algorithm by transforming a DTD and set of XFDs into new DTD that is in XNF. The input of this algorithm is a DTD and XFD and the output of this algorithm is a new DTD that is in XNF which contains the same information. This algorithm consists of two rules: create new element types and moving an attribute. Using this algorithm, the following new DTD will be derived after eliminating XFD in (*1*). The algorithm will be repeatedly applies until the DTD is in XNF.

Figure 6: New DTD after elimating XFD *(1)*

```
<!DOCTYPE department[
<!ELEMENT department(course*, tinfo*)>
     <!ELEMENT course(title, student*)>
       <!ATTLIST course cno ID #REQUIRED>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT student(fname|lname?,lecturer)>
        <!ATTLIST student  Sno ID #REQUIRED>
     <!ELEMENT fname(#PCDATA) >
    <!ELEMENT lname(#PCDATA) >
    <!ELEMENT lecturer(EMPTY)
      <!ATTLIST lecturer  tno ID #REQUIRED>
    <!ELEMENT tinfo (tname)>
        <!ATTLIST lecturer  tno ID #REQUIRED>
       <!ELEMENT tname (#PCDATA)>
]>
```

2.3. Discussion

Both approaches have a remarkable impact on XML document design. Each of them is complemented by the other in XML design research. Arenas and Libkin[1] have defined XFDs and XML normal forms (XNF) entirely within the context of the XML document. XFD is formally defined based on the concept of 'tree tuple'. Arenas and Libkin[1] proved that their XNF can avoid redundancies and update anomalies using information theory measure [1] at the schema and instance levels. They also showed that XNF is generalised from BCNF if the XML schema is converted into a relational presentation. However, the problem with this approach is that the way they express the semantic constraint (functional dependency) which is very complicated due to the textual presentation of a schema. As we know, functional dependency is already the area where designers have the most problem specifying in relational models, so making them more complicated and unfamiliar to designers make XML document design more difficult. Moreover, a common problem with this approach is that the whole schema has to be redesigned when requirements change and information is added or withdrawn. In addition, the functional dependencies defined by Arenas and Libkin [1] are dependent on the XML labelled tree (a model for XML documents) where paths are defined through the tree. Therefore when paths change, the functional dependency is adjusted as well. For this reason XNF can never be dependency preserving [11]. Another shortcoming is that both DTD and XML tree are represented in textual representation, and so as a result it is difficult to visualise the data and their relationship.

In contrast, an ORA-SS data model is proposed to assist in XML document design [6] at the conceptual level. The hierarchical structure of the object class is clearly shown in ORA-SS model. In Dobbie et al's work, it was assumed that the starting point for the design of an XML document is at a

conceptual model ORA-SS. Using the algorithm, then an XML document is derived from a normal form ORA-SS [15]. This approach followed from the ER normal form [15]. Embley and Ling[3] and Ling [15] have proved that this approach is guaranteed to produce a redundancy-free and compact relational database. In traditional database design, practitioners routinely use an ER model and convert the ER diagram to a relational model. Another advantage of this approach is that it is easier and simpler for designers compared to normalization theory [15,9]. However, the normal form ORA-SS relies upon definition of NF-NR [13]. In order to use and understand the normal ORA-SS the user must understand normal form for nested relations first.

The major difference between the above work and our work is that we address several issues that were not considered in Ling et al [15] and Arenas & Libkin [1]. Firstly, we propose and add some new notations to our model GN-DTD to express explicitly more semantics of the XML document. Secondly, in contrast with Ling et al's approach, we assume the XML document must be associated with DTD; hence extraction of the schema from the XML document is not required. Thirdly, we propose multilevel normal form for GN-DTD by considering different types of data dependency such as relationship dependency, partial dependency, transitive dependency and global dependency. These data dependencies are presented in a simpler way in order to describe the semantics of real data. To form this normal form we adopt some rules in Arenas and Libkin[1] and Ling et al [15]. Finally, an algorithm is developed for converting an un-normalized GN-DTD to a normalized one by avoiding reference relationships and preserving the data dependency as well. This is important because other than no redundancy, data preservation is a fundamental criterion in determining the quality of DTD as it preserves the structural and semantic information of the XML document entirely.

3. The GN-DTD

Some of the notations of GN-DTD have been adopted and improved from the current data model ORA-SS [6] notations and conventional ER model [4]. The significant difference between GN-DTD and the other models is complex element, simple element, and attribute nodes are explicitly distinguished. This is because such distinction is crucial for clarifying the semantics of the data. More importantly, GN-DTD provides an explicit semantic relationships definition to allow the user to define semantic relationship between nodes. The GN-DTD allows users to define the structure of nodes in a hierarchical way. We have made the ordering of sub-elements significant by treating them as a set, sequence (list) or disjunction. The GN-DTD model is briefly discussed in the following section. The details of the GN-DTD specification can be found in [25].

3.1. Syntax and Semantics of GN-DTD

Graphical Notation for Data Type Definition (GN-DTD) graphical modeling [25] consists of six basic components:

- *Complex element nodes. A* complex element node is used to represent a set of elements which have another sub element and attribute. The complex element node is illustrated as labeled rectangle box. This notation is adopted from the ER model [4] which is similar to entity. The label is written in the rectangle box as a tuple <*name,level*>, where *name* represents the name of the node and *level* represent the depth of the node in GN-DTD. The name is mandatory.
- *Simple element nodes.* A simple element node is used to represent an element associated with #PCDATA or #CDATA. It is illustrated as a labeled rounded rectangle box with the form <*name,level,type*> where *name* is the name of the simple element, *level* is the depth of the node in the GN-DTD and *type* represents PCDATA or CDATA or string 'S' . All simple element nodes are assumed to be mandatory and single valued, unless the node contains the symbol ? which signifies it is single value and optional, or + which signifies that it is multi-valued and required, or an * which shows that it is optional and multi-valued. This notation is similar to

ORA-SS [4]. The symbol is written in front of the tuple <*name,level,type* > to differentiate among them accordingly.

- *Attribute nodes.* Attribute nodes are used to represent attributes defined in *ATTLIST*, which describe the properties of a complex element node. An attribute node is an identifier for a complex element node. It is represented as ID which is unique and mandatory among the instances of complex element. Attributes can be classified as single attributes and composite attributes. A single identifier attribute has an atomic value and composite attributes have more than one identifier attributes. A single identifier attribute is represented as an oval and a composite attribute as a double oval.
- *Set relationship type.* Two types of link exist in GN-DTD: Inheritance link and Part- of Link

(i) The *inheritance link* is a relationship between a complex element node and another complex element node. This link shows inheritance between parent nodes to child nodes or ancestor node to descendant node. For inheritance relationships, a semantic meaning, which is indicated by the *connectivity* between complex element occurrences is important. The connectivity of a relationship specifies the mapping of the associated complex element occurrence in a relationship. Basic constructs for connectivity are: *one to one* (unary or binary relationship), *one to many* (unary or binary relationship), and *many to one*, *many-to-many* (unary or binary relationship). All these type of relationships are indicated by directional arrows. To differentiate among them, both cardinality constraint and degree are attached to the arrow. The notation as presented as (*name, d, cp,cc)* where *name* represents the name of the relationship, *d* is the degree of relationship, *cp* and *cc* are cardinality constraints for parent and child respectively. This notation is similar to ORA-SS [15].

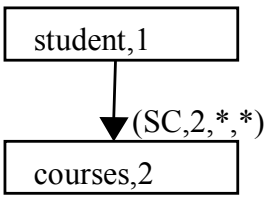
(a) Degree of Relationship

Degree of relationship is the number of complex elements associated with the relationship. An n-ary relationship is of degree n. Unary, binary and ternary relationships are special cases in which the degree is 1, 2, and 3, respectively.

(b) Cardinality Constraint for Complex Element

To reveal more semantics in their relationship, the cardinality constraint is associated with the inheritance link. The same rule applies for cardinality constraints for both parent node (cp) and child node (cc). Here cardinality of complex elements in a relationship is represented as a 2 tuple (min:max). The constraint *(*0:N), (0:1)and (1:N) is represent as the operators *, ? and + respectively except cardinality constraint (1:1) is presented as 1. This relationship cardinality constraint is indicated using a directional arrow: For instance, the diagram in Figure 8 illustrates a binary inheritance relationship between complex element student and complex element courses where a student can take zero or many courses while many courses can be taken by zero or many students.

Figure 8: Many-to-many Binary Relationship



(ii) Part-of Link

Part-of link is divided into two types: Part-of link attribute is a relationship between a complex element node and attribute node. This is a mandatory relationship. It is illustrated as a bold double arrow. Part-

of link simple element is a relationship between a complex element node and a simple element node. This link can be optional. It is illustrated as a single double arrow. For instance, attribute identifier for a complex element is linked as a one-to-one relationship.

- *Semantic constraint between set relationships*. There are two types of set relationships: First, sequence between a set of child elements nodes. We emphasise in our notation that the attribute node(s) must be located in the first position in the sequence. For instance, the notation *sequence* is used to capture the list of the first author and the second author of a book. To express such ordering in a GN-DTD, we draw a directed upwardly curving arrow labelled with {sequence} across all the set of relationships involved. Second, disjunction between the set of sibling nodes. To illustrate this, we draw a line labelled with {XOR} across all the set of relationships involved.
- *A root node.* Root node notation is similar to complex element notation where it is a special case of a complex element node and its level is always zero.
- *A last node.* A last node notation is similar to attribute and simple elements notation where their level is at n-1.

All notations for GN-DTD decsribed above are shown in Figure 9.

Figure 9: GN-DTD's notation for syntax and semantics

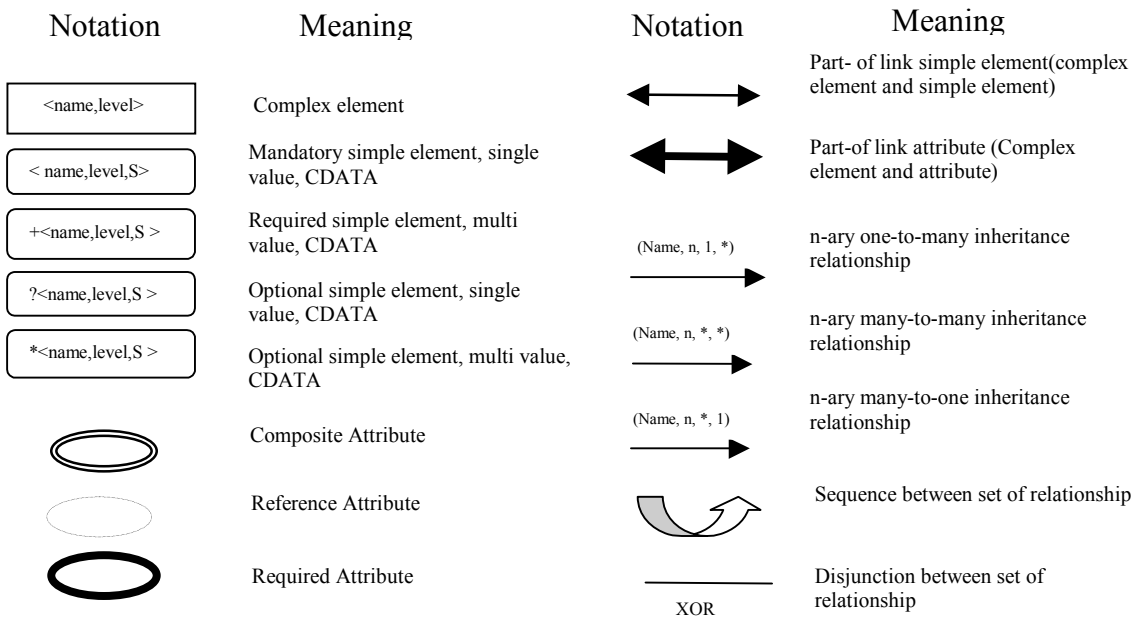
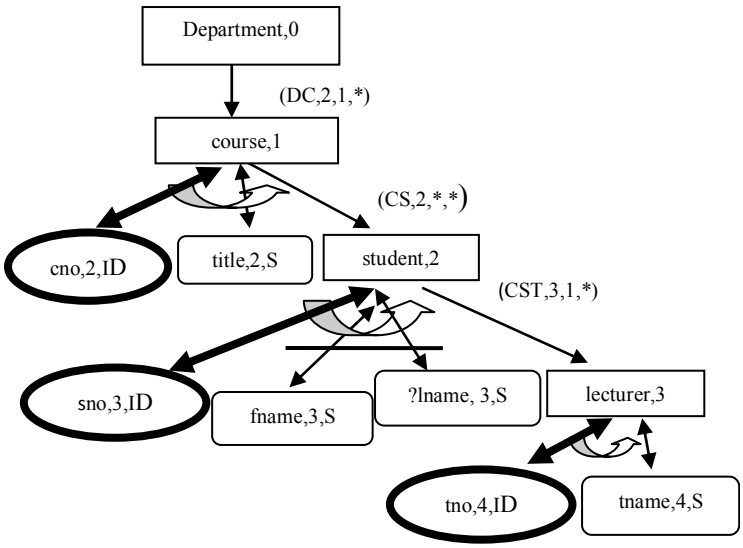


Figure 10 shows GN-DTD describing the structure of XML document corresponding to DTD in Figure 10 shows GN-DTD describing the structure of an XML document corresponding to DTD in Figure 1. Root node *Department* has a binary inheritance relationship with the complex element node *course*. The semantic relationship between them reveals that the *Department* can have one-to-many *courses* at one time. The complex element *course* has a sequence of attribute *cno,* simple element node *title* and complex element node *student*. The part-of link attribute is a mandatory relationship where the attribute node *cno* is required and unique for every course in the XML document. The simple element node *title* is part-of the complex element courses. One *course* can be taken by many *students* while the complex element *student* consists of a sequence of attribute node *sno*, simple elements fname, *lname* and complex element *lecturer.* Attribute node *sno* is required for the complex element *student*. Complex element node *student* requires only one of its subelements either *fname* or *lname* to appear in the XML document while the simple element *lname* is optional. The semantic relationship between course, student and lecturer is indicated as a ternary relationship since each student is assigned to a lecturer who is teaching the course. Attribute *tno* is required while simple element *tname* is

mandatory and string S denotes a node is a type PCDTA. Attribute key *tno* and simple element *tname* are the last node in GN-DTD. The level of each node is indicated explicitly in the model.

Figure 10: GN-DTD



3.2. Data Dependencies of GN-DTD

Data dependencies are part of the real world semantics [14, 1]. They represent the semantic information in the form of relationships between different attributes in the XML documents. In Ling [14], it is stated that "*data dependencies should be modelled precisely early in the design stage for a correct and complete database representation of semantic*." In a database context, data dependency can be categorized into functional dependency and key dependency.

3.2.1. Key Attribute

We define a key attribute as a unique attribute that can determine uniquely other simple element in the whole XML document. For instance course number (cno), student number (sno) and lecturer number (tno) are unique and mandatory, because they are represented as one-to-one relationships between complex element and attribute node. We define this in GN-DTD as a key attribute similar to [3].

3.2.2. Functional Dependency

Functional dependency models real world constraints, showing that some of the complex elements or simple elements/attributes depend on other complex elements or attributes/simple elements. Based on the GN-DTD model, we define four types of functional dependencies: relationship dependency, global dependency, transitive dependency and partial dependency

3.2.2.1. Relationship Dependency

The relationship dependencies are presented clearly in the GN-DTD diagram using a directed arrow. Our XML relationship dependency is defined in terms of a uniqueness constraint in the relationship between complex element nodes GN-DTD. These types of relationships could cause data redundancy and should be eliminated from GN-DTD. The categories are:

- *one-to-many binary inheritance relationship dependency*
- *many-to-many binary inheritance relationship dependency*
- *many-to-one binary or ternary inheritance relationship dependency*
- *many-to-many ternary inheritance relationship dependency*

For instance, the XML document illustrated in Figure 3 shows that *student* name "David Grey" and both *lecturer* name "Bing" and "Bottaci" have appeared twice in the document. This is because *one-to-many* ternary inheritance relationship between course, student and lecturer exist in GN-DTD as shown in Figure 10. Therefore to achieve a redundancy-free XML document, we need to avoid these types of relationship dependencies in GN-DTD.

3.2.2.2. Global Functional Dependency

Global functional dependency occurs based on a few reasonable constraints (dependency) that a database designer may specify for his/her application. Global functional dependency holds in GN-DTD if there exists a dependency between the attribute and simple element of the corresponding complex element under a binary or n-ary many-to-many inheritance relationship at the last node of GN-DTD. We adopted this definition from [1, 15] with some modification to suit the hierarchical structure of XML documents.

Definition 1:

Let CE be a set of complex elements under binary or n-ary one to many/many-to-many inheritance relationship, Let ATT be an identifier attribute of CE and SE is a simple element of CE. The global functional dependency of GN-DTD is defined as follows:

(1) For each part-of-link simple element <SE,l_{n-1}> of <CE,l_{n-2}>, where SE and ATT are last nodes in GN-DTD

<ATT,l_{n-1}>→<SE,l_{n-1}> is a global dependency in GN-DTD. SE can be a set of simple element, list of simple elements or single value of a simple element.

(2) For each last node attribute ATT1 of CE which is not the attribute identifier of CE, ATT→ATT_1 and ATT_1→ATT is also a global dependency.

For instance, in Figure 10, the possible global dependency (GFD) of the complex element lecturer is as follows:

tno→ tname (2)

This above GFD *(2)* represents the constraint, whenever two nodes agree on the value of all attributes *tno*, they also agree on the value of all attributes in *tname*. This GFD can be viewed as a function from one set of attributes/simple elements to another set of attributes/simple elements. This GFD is similar to XFD defined by Arenas and Libkin[1] but we put it in a different way without using the path notation.

3.2.2.3. Functional Transitive Dependency

We adopted the definition of transitive dependence from the relation model [5] and nested relation [14]. Transitive functional dependencies between complex elements occur if theirs attribute or simple element node has dependency with another simple element node from a different level.

Definition 2:

Case 1: Let ATT be a key identifier for complex element(CE) and {SE_a, SE_b} are simple elements for CE

If there exist two constraints <ATT,l_i> →<SE_a,l_i> and <SE_a,l_i →<SE_b,l_i>, then we say that attribute <SE_b,l_i> is transitively dependent on <ATT,l_i>

Case 2: Let ATT_a be a key identifier for CE_a , ATT_b is a key identifier to CE_b and SE is a simple element for CE_c and they are located in different levels.

If there exist two constraints <ATT_a,l_i> →<ATT_b,l_{i+1}> and <ATT_b,l_{i+1}>→ <SE, l_{i+1}>, then we say that <SE, l_{i+1}> is transitively dependent on <ATT,l_i>

3.2.2.4. Functional Partial Dependency

We adopted the definition of functional partial dependency (PFD) from [17].

Definition 3:

Let ATT_a,ATT_b, ATT_m be a key identifier for CE_a,CE_b and CE_m respectively. These CE have a binary or tenary relationship with each other and are located in different levels.

If there exist two constraints $PFD1: \{<ATT_a,l_i>, <ATT_b,l_{i+1}>,<ATT_m,l_{i+2}>\} \rightarrow <ATT_n,l_{i+3}>$

$PFD2 : <ATT_a,li> \rightarrow <ATT_n,l_{i+3}>$

then constraint *PFD2* is called functional partial dependency because it is a subset of *PFD1* where attribute $<ATT_a,l_i>$ alone can be used to determine $<ATT_n,l_{i+3}>$

Functional partial dependencies involve composite attribute keys. The composite key attribute could be from the same level or from a different level. The subset of composite key attribute can functionally determine the simple element node.

4. Normal Form for GN-DTD

We next define multilevel normal forms for GN-DTD based on the notion of data dependencies and relationship dependencies of GN-DTD.

4.1. First Normal form GN-DTD (1XNF GN-DTD)

The first normal form for GN-DTD is about finding unique identifier attributes for the complex elements set, and checking that no node (complex element, simple element or attribute) actually represents multiple values. To be in first normal form, each attribute, complex element or simple element is not NULL and has a single label. More importantly, the primary key (unique identifier) for the complex element must be defined. To be precise, we propose the following rules.

GN-DTD is in first normal if and only if:

a) Only one value for each simple element node or attribute node of GN-DTD can be stored. If there is more than one value, we must add some new element nodes or attribute nodes to store them. For instance, consider Figure 10. If the complex element *course* has two titles, we need two *title* simple elements node for each *courses* to store the two *title* names. This is equivalent to being 'no repeating group' in relational schema [5].
b) The root element of a GN-DTD model should be located at level 0 and the cardinality of the root element node must be one.
c) Each set of complex element node in the GN-DTD has at least one key attribute node.

4.2. Second Normal form GN-DTD (2XNF GN-DTD)

Some nodes need to be restructured. However they can then still be in a single GN-DTD. This is possible in XML because XML supports hierarchies in a single document, while relational databases do not support hierarchies in a single row. This is different from the relational second normal form (2NF), which requires one-to-many relationships to be in separate tables.

The GN-DTD is in second normal form if and only if:

a) GN - DTD is in 1XNF.
b) There is no nested binary inheritance relationship or ternary inheritance relationship under many-to-many or one -to-much inheritance relationships with the following condition:

For each nested set of complex element<CE,*l+1*> of <CE,*l*>, and any key attribute (ATT) of <CE,*l*>, the key attribute and simple element of <CE,*l+1*> is not partial dependent on ATT of complex element<CE,*l*>

4.3. Third Normal form GN-DTD (3XNF GN-DTD)

In the third normal form of GN-DTD, making a change to one unique complex element node set would not affect the integrity of another complex element node set. If needed, a complex element node set would be divided into two separate complex element node sets. This is similar to the relational 3NF.

GN- DTD is in third normal form if and only if:

b) GN-DTD is in 2XNF.
c) There exists no nested inheritance relationship type of *n*-ary many-to-one or many-to-many under a one-to-many inheritance relationship set in GN-DTD and the following conditions are satisfied:
   (i) For each nested set of complex elements<CE_b,*l+1*> of set of complex element<CE_a,*l*>, any key attribute and simple element of <CE_b,*l+1*> is **not transitively dependent** on ATT of complex element<CE_a,*l*>
   (ii) Any key attribute node of any complex element node located in a different level are disjoint (ATT<*CE,l*> ∩ ATT<*CE,l+1*>∩ ATT<*CE,n*> =0)

4.4. Normal form GN-DTD (NF GN-DTD)

GN- DTD is in Normal Form if and only if:

a) GN-DTD is in 3NF.
b) There are no global dependencies between attribute and simple element of complex element nodes under nested one-to-many or many-to-many inheritance relationships

5. Transformation from Un-normal Form GN-DTD to the Normal Form GN-DTD

The common feature of normalization procedure is to convert an initial schema into one in a normal form to reduce anomalies and redundancies in the XML document. In this section, we propose rules to transform the un-normal form GN-DTD into a normal form one. These rules are used to remove redundancy in XML documents caused by relationship dependency, global functional dependency, partial functional and transitive functional dependency. Generally, in these rules, we first restructure the GN-DTD by creating a new complex element node, moving up node and moving sub-tree node. We subsequently adjust the semantic relationship between simple element nodes or attribute nodes in GN-DTD. We next define the normalization rules.

5.1. Normalization Rules

The following notations will be used in the following rules.

r	*represents root element*
ATT	*represents attribute*
SE	*represents simple element*
CE	*represents set of complex elements*
$\rightarrow^R$	*represents relationship*
l	*represents level of node where (0 ≤ l ≤ n-1) , n is a finite positive number*

Rule 1: Eliminate Redundancy though Global Functional Dependencies

Let *ATT→{SE} for Complex element CE*

1.1 Create a new set of complex element name <*CE _new,l*>

1.2 Locate node <*CE_new,l*> at level one (*l=1*) at the rightmost position of GN-DTD

1.3 Create new relationship type of binary one-to-many binary *inheritance link* between root and new set complex element name <*CE_new,l*>

1.4 Replicate attribute node *ATT* and simple node *SE*

1.4.1 Make them as a children to node <*CE_new,l*>

1.4.2 Let new attribute node (*ATT)* of <*CE_new, l*> be a key node

1.5 Create a new relationship type of *part-of link* between attribute node *ATT* and simple element node *SE* with <*CE-new, l*>

1.6 Delete simple element node {*SE}* from original location and its relationship

1.7. Eliminate global dependency *(1)* from set of data dependency

Rule 2: Eliminate n-ary many-to-many or one-or-many inheritance relationship with partial transitive

To eliminate n-ary many-to-many or **one-or-many** relationship we must avoid multi – hierarchy. For each, n-ary relationship *R* (n>2), many-to-many or one-to-many relationship type

(<$CE_{a,}$, l > $\rightarrow^{R}$<CE_b, $l+1$>)

4.1 Create a new set complex element name <CE_b _new,l>

4.2 Locate node <CE_b_new,l> at with the same level of level <$CE_{a,}$, l > at the rightmost position of GN-DTD

4.3 Create a new relationship type of binary one-to-many binary *inheritance link* between parent of <$CE_{a,}$, l > and new set complex element name <CE_b_new,l>

4.4 If partial dependency exists in the <CE_b, $l+1$> then replicate all children of <CE_b, $l+1$> to be children of the set complex element <CE_b_new,l>

4.5 Delete all children of <CE_b, $l+1$> except the attribute key node

Rule 3: Eliminate binary many-to-many/many-to-one/one-to-many Inheritance Relationship with Transitive Dependency

For each, binary many-to-one relationship type

(<$CE_{a,}$,l > $\rightarrow^{R}$<CE_b, $l+1$>)

If exist, <CE_a,l> *with attribute* ATT_a *and* <CE_b,$l+1$> *with attribute* ATT_b, *with and simple element* SE_b

Where $ATT_a \rightarrow ATT_b$

$ATT_b \rightarrow SE_b$

$ATT_a \rightarrow SE_b$ *is transitive dependency*

3.1 Move up the set complex element <CE_b, $l+1$> along with its children to the same level <$CE_{a,}$,l >

3.2 If parent of <$CE_{a,}$,l > is a root node

then create a new relationship type of one-to-many *inheritance link* between parent of <CE_a,l> with set complex element node < CE_b,l+1 >

else create new relationship type of many-to-many *inheritance link* between parent of <$CE_{a,}$,l >

3.3 Eliminate transitive dependency ATTa $\rightarrow$ SE_b from set of complex element <$CE_{a,}$,l > and <CE_b, $l+1$>

5.2. Normalization Algorithm

Normalization is a process that analyses and restructures the schema of an XML document to minimize redundancies with the help of data dependencies in the data. The normalization algorithm takes GN-DTD and set of data dependencies specified by the user as an input, and returns the normal form GN-DTD as output. These algorithms apply to normalization rules presented in the previous section. We propose three algorithms to transform a GN-DTD into 2XNF, 3XNF and XNF, respectively.

5.2.1. Algorithm 2XNF GN-DTD

Input: The GN-DTD D in 1NF-DTD and set of specified dependencies constraint.
Output: The GN- DTD in 2NF-DTD
Let D be GN-DTD
If D is 2NF GN-DTD, then output D
Search all inheritance relationships in GN-DTD from root to last node
If there exist **n-ary many-to-many inheritance relationship** or **one-to-many inheritance relationship** under
many-to-many inheritance relationship
Then take the last inheritance relationship in the GN-DTD
Restructure D by applying rule 2
Repeat until no more partial functional dependency
Output D

5.2.2. Algorithm 3XNF GN-DTD

Input: A GN-DTD D in 2NF GN-DTD and given set of data dependencies.
Output: A GN-DTD in 3NF GN-DTD
Let D be DTD
If D is 3NF GN-DTD, then output D
else
If there exist many-to-many or many-to-one or one-to-many inheritance relationship with transitive functional dependency
then restructure D by applying rule 3
Repeat until no more transitive functional dependency
Output D

5.2.3. Algorithm XNF GN-DTD

Input: A GN-DTD D in 3NF GN-DTD and given set of data dependencies.
Output: A GN-DTD in NF GN-DTD
Let D be GN-DTD
If D is XNF GN-DTD, then output D
Get all the dependency constraint
If there exist constraint with global functional dependency
then apply rule 1
Repeat until no more global functional dependency
Output D

5.3. Examples

In this section, we present examples to illustrate how the GN-DTD is restructured according to 1XNF, 2XNF, 3XNF and XNF respectively. We assume a set of functional dependencies presented in the following section with the GN-DTD in Figure 10 are used as input.

5.3.1. Example of INF GN-DTD

The following constraints are given based on database designer's requirements:

Constraint 1: *Each set course, student and lecturer has unique identifier.*
Constraint 2: *student number (sno) → {fname, lname}*
Constraint 3: *course number (cno)and student number(sno) → lecturer number (tno)*
Constraint 4: **course number(cno) → lecturer number(tno)**
Constraint 5: **course number(cno) and lecturer number (tno) → lecturer name(tname).**
Constraint 6: *lecturer number(tno) → lecturer name(tname)*

The GN-DTD illustrated in Figure 10 shows that it is in first normal form (1 XNF) because each set of complex element nodes *course*, *student* and *lecturer* has *cno, sno* and *tno* as a unique key identifier respectively, while all simple element nodes and attribute nodes have one unique label. The *department* node is a root element since it is located at level 0. Moreover, the XML documents shown in Figure 2 satisfy and conform to GN-DTD in Figure 10.

5.3.2. Example of 2XNF GN-DTD

GN- DTD of Figure 10 is not in 2XNF because

- There exists a ternary inheritance relationship type nested with many-to-many inheritance relationship type. This dependency relationship involves *course*, *student* and *lecturer* nodes. As a consequence information about *lecturer* is stored redundantly in the XML document and can cause update anomaly. If the information about the *lecturer* is changed, then it must be updated in all subtree *students* who are taking the same *course.*
- There exists a partial dependency between complex elements *course, student* and *lecturer.* This partial dependency is caused by constraint 3 and constraint 4.

To be in 2XNF GN-DTD, the new set complex element *lecturernew* along with its children is created at the same level as complex element *student* node. A many-to-many inheritance link between *course* and *lecturernew node* is created. All children from the original complex element *lecturer* are deleted except key attribute node *tno*. In this way, the original semantic relationship is preserved in GN-DTD. Figure 11 presents 2XNF GN-DTD. Having this structure, partial dependency is eliminated in complex element lecturer as well, but still preserved in *lecturernew*.

5.3.3. Example of 3NF GN-DTD

GN- DTD of Figure 11 is not in 3XNF because

- There exists many-to-many inheritance relationships between element *lecturernew* and *course* node under one-to-many inheritance relationship.
- There exists transitive dependency between complex element *course* and complex element *lecturernew* with *cno→tname*. This transitive dependency is derived from constraint 4 and constraint 6.

To be in 3NF GN-DTD, the inheritance link between the set of complex element node *course* and complex element *lecturernew* node needs to be restructured and transitive dependency eliminated within course node and complex element *lecturernew*. The set of complex element node *lecturernew* along with its children is moved up and linked with department node. Because department is root, a binary one-to-many hierarchical link is created. Figure 12 presents a new structure of 3XNF GN-DTD after eliminating the above constraints.

5.3.4. Example of XNF GN-DTD

GN- DTD of Figure 12 is not in XNF because

(i) There exists global dependency sno *→{fname,lname}* for set complex element *student* node and *course* node under many-to-many relationship. If the information about the *fname and lname* is changed, then it must be updated in the whole document.

To remove redundancies caused by global dependencies, a new *set* of complex element node *studentnew* is created at level one. A new binary one-to-many *inheritance relationship* between *studentnew* and *department* node is created. The attribute node *sno* and simple element node *fname* and *lname* are replicated and they become children of set complex element *studentnew*. The *part of link* is created between them accordingly. Both of simple element node *fname and lname* are deleted from *student* node but key attribute node *sno* is remained as a child for complex element *student* node. Figure 13 illustrates new structure of XNF GN-DTD with free redundancy. The result shown in Figure

13 is similar to the XNF defined by Arenas and Libkin[1]. As part of the design process shown in Figure 3, finally, XNF GN-DTD will be mapped back to the new DTD by applying transformation rules. We present these transformation rules next.

Figure 11: GNT-DTD in second normal form

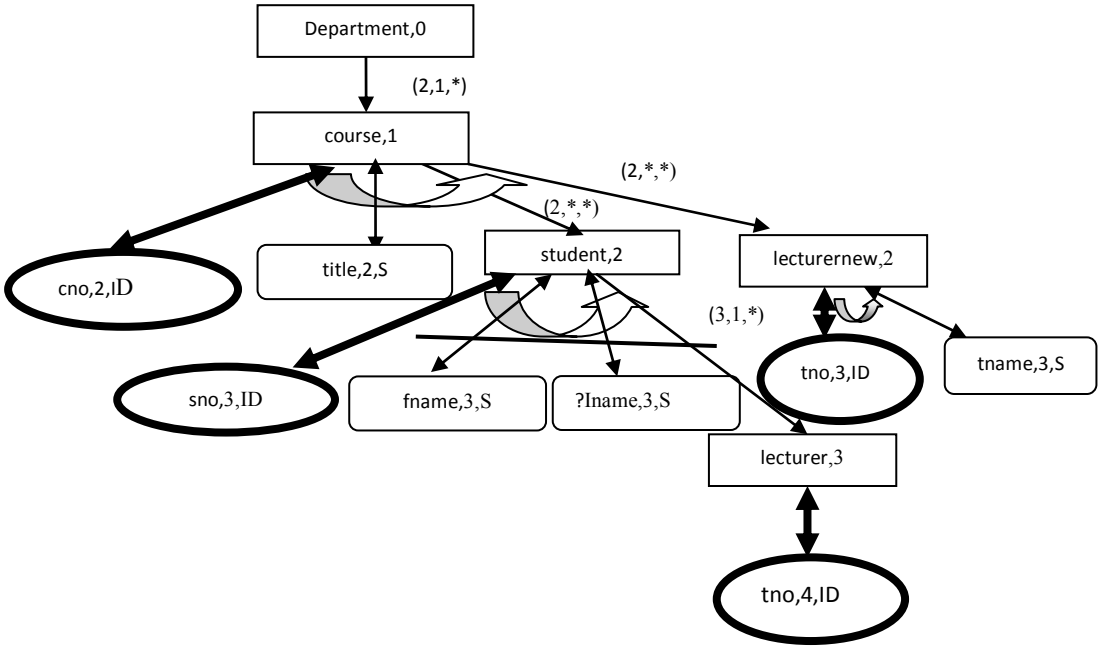


Figure 12: GN-DTD in third normal form

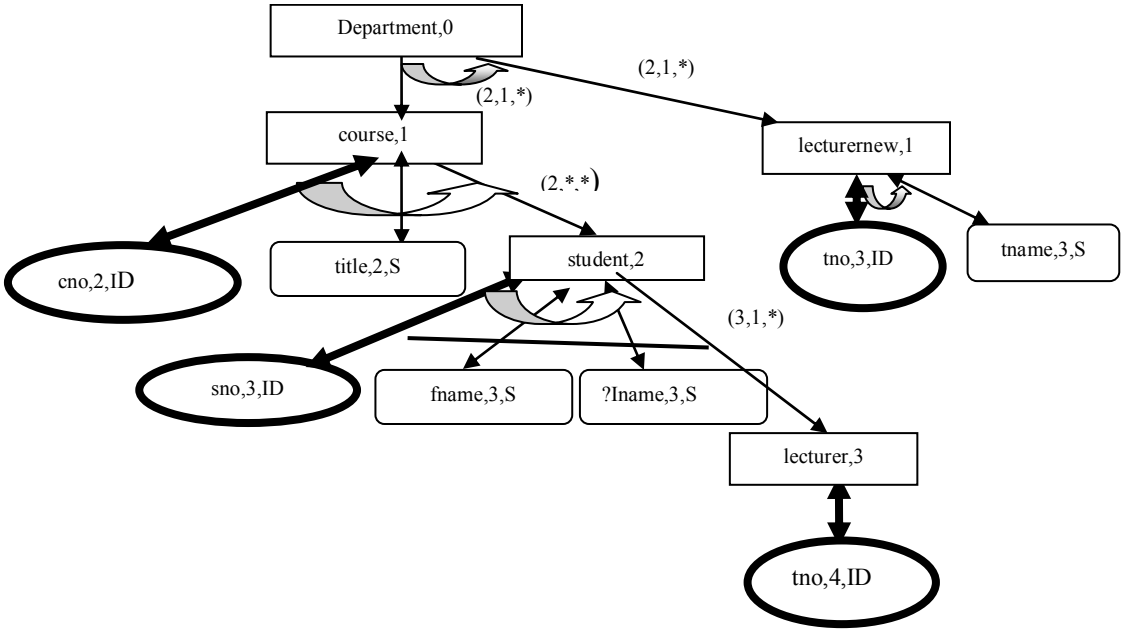
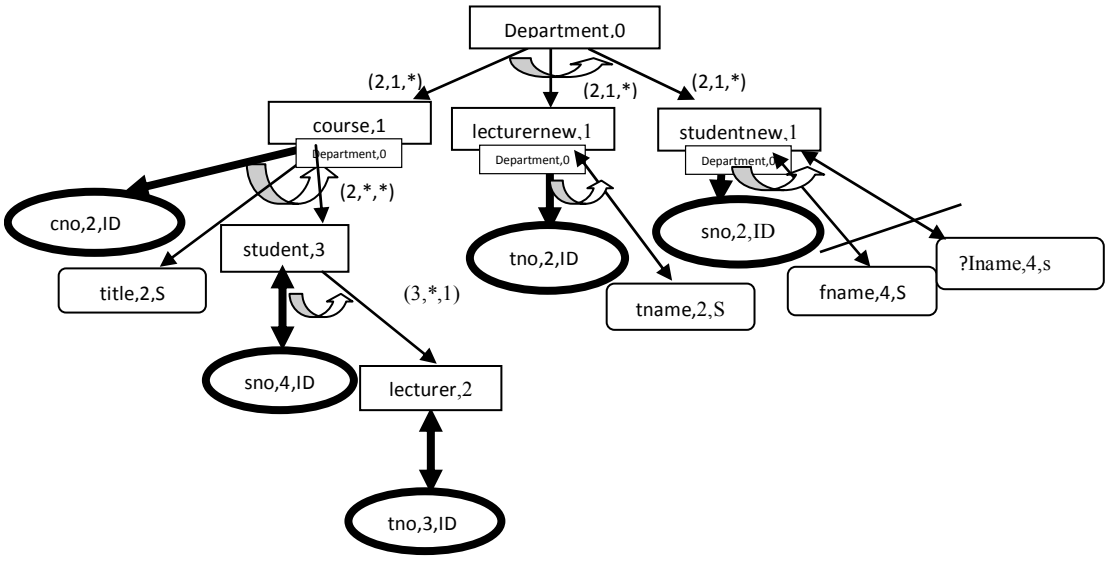


Figure 13: GN-DTD in normal form



6. Transformation Rules

Given the GN-DTD, the syntax and structure of DTD can be derived easily. This can be achieved by traversing all the nodes from one level to another level starting from the root node up to the leaves nodes.

Step 1 Level 0, a root node is represented by <!DOCTYPE *root node name [element type definition]* >

Step 2 Level 1, identify the subtree of GN-DTD, check the number of nodes, type of nodes and relationship type

Step 3 If there is more than one node at level 1 and the relationship type between root and child node(s) is a binary one-to-many inheritance relationship then generate

<!ELEMENT *root node name* (*Ni*))>

Where *Ni* is the list of subelements/child nodes

3.1 Certify the relationship set between *parent* nodes and *child* nodes,

3.1.1 If {XOR} means the relationship between node is a disjunction and will be represented using symbol '|'

Else

3.1.2 If {sequence} means the relationship is sequence and will be represented using symbol ','

3.2 Verify the semantic constraint between complex element nodes (*parent)* and *complex element* nodes (child) in each of relationship set and map to the following operator:

3.2.1 if (m,1,*) or (m,*,*) or (m,*,1) map to operator *

3.2.2 if (m,0,*) map to operator +

3.2.3 if (m,0,1) map to operator ?

Where *m* is n-ary relationship and n > 1

Step 4 If the list of subelements (*Ni*) is not empty, using depth first traversal, for each node in list subelement *Ni*

4.1 generate <! ELEMENT *Ni* (subelement *Nj*)>

4.2 repeat step 3.1 and 3.2

4.3 for each complex element (*Ni*), if the relationship between them is part-of link attribute (one-to-one) then generate

<! ATTLIST *Ni* attribute name ID # REQUIRED>

4.4 For subelement *Nj*

4.4.1 If *Nj* is a simple element has part of link simple element (many-to-one or one-to-many relationship) with *Ni* then generate

<! ELEMENT simple element name #PCDATA>

(Repeat for all simple element nodes)

4.4.2 If *Nj* is a complex element node has inheritance link with complex element Ni

Repeat step 4

4.4.3 If *Nj* is a complex element node has part of link then generate

<! ELEMENT *Nj* (EMPTY) >

Step 5 Go to next subtree GN-DTD and repeat step 4

Using transformation rules, GN-DTD in Figure 13 is mapped to a new DTD and finally initial XML documents is restructured to conform to this new DTD as shown in Figure 14 and 15 respectively.

Figure 14: New DTD

```
<!DOCTYPE department[
    <!ELEMENT department(course* , lecturernew*, studentnew*)>
     <!ELEMENT course(title, student*)>
       <!ATTLIST course cno ID #REQUIRED>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT student (lecturer)*>
        <!ATTLIST student  Sno ID #REQUIRED>
    <!ELEMENT lecturer *(EMPTY) >
        <!ATTLIST lecturer  tno ID #REQUIRED>
    <!ELEMENT lecturernew *(tname)>
      <!ATTLIST lecturernew  tno ID #REQUIRED>
    <!ELEMENT studentnew *(fname , lname) >
        <!ATTLIST studentnew  Sno ID #REQUIRED>
   <!ELEMENT fname(#PCDATA) >
    <!ELEMENT lname(#PCDATA) >
]>
```

Figure 15: Non redundancy XML document conforms to DTD in Figure 14

```
<!DOCTYPE Department [
  <course>
     <course cno = "csc101">
     < title > XML database </title>
      < student >
          <student sno = "112344">
         <lecturer>
          <lecturer tno = "123">
        </lecturer>
      </ student >
      < student >
          <student sno = "112345">
       <lecturer>
         <lecturer tno = "123">
      </lecturer>
   </student >
 </course>
<course>
    <course cno = "csc201">
    < title > Database technique </title>
   < takenby>
      < student >
        <student sno = "112344">
       <lecturer>
         <lecturer tno = "123">
       </lecturer
</ student >
      < student >
        <student sno = "112346">
       <lecturer>
         <lecturer tno = "123">
       </lecturer
      </student >
</course>
<course>
    <course cno = "csc102">
    < title > Z formal methods </title>
     < student >
         <student sno = "112344">
        <lecturer>
          <lecturer  tno = "124">
       </lecturer>
     </student>
       < student >
          <student sno = "112345">
        <lecturer>
           <lecturer  tno = "124">
        </lecturer>
      </student>
    </course>
<lecturernew>
     <tinfo tno = "123">
     <tname> bing </tname>
</lecturernew>
<lecturerinfo>
      <tinfo tno = "124">
     <tname> bottaci</tname>
</lecturernew>
<studentnew>
       <studentnew sno = "112344">
        <fname> David</fname>
        <lname> Grey </lname>
</studentnew>
<studentnew>
       <studentnew sno = "112345">
        <fname>Helen</fname>
</studentnew>
</Department>]
```

7. Conclusion and Future Work

We have proposed a method for designing a "good" XML document in two steps: first, we building a conceptual model by means of GN-DTD at the schema level and second, using normalization theory where functional dependencies are refined among its simple elements and attributes. The GN-DTD can be further normalised either to 1XNF, 2XNF, 3XNF or XNF using the proposed normalization algorithm. In the proposed methodology, a GN-DTD is used as input and the normalization rules are applied during the normalization process. In this process, the original GN-DTD is restructured accordingly by considering the hierarchical relationship types, the level of nodes, and the set of data dependencies given between nodes. The normal forms GN-DTD presented in this paper have shown three advantages. First, the designer can indentify complex elements, simple elements, and attributes graphically and can add the relationship types between the nodes from the user specification. This will give more control to the designer to evaluate each normal form GN-DTD. Second, GN-DTD is able to

preserve both DTD hierarchical structure and XML document structure and satisfy user requirements. More importantly, the semantic constraint in the GN-DTD is preserved. Although we have chosen DTD rather than XML schema as a starting point to research XML normalization, the concept and methods used in this work can be generalised to XML schema. As an ongoing work, we are implementing this normalization process using a formal specification method.

References

[1] Arenas, M. and Libkin, L. A Normal Form for XML Documents, *ACM Transaction on Database System*,vol 29(1), 2004, pp. 195-232.

[2] Bird,L., Goodchild, A., and Halpin, A. Object Role Modeling and XML-Schema. ER 2002, pp. 309-322.

[3] Buneman, P., Fan,W.,Simeon, J., and Wienstein,S. Constraints for Semistructured data and XML. *SIGMOD record 30* , 2001, pp. 47-54.

[4] Chen, P.P. The entity-relational model: Towards a unified view of data, *ACM transaction on Database System*, 14, 1976.

[5] Codd, E. Further Normalization of the DataBase Relational Model. In *Database system, Computer Science Symposia series 6.* Prentice Hall,1972.

[6] Dobbie, G., Xiaoying, W., Ling. T.W., and Lee, M.L. ORA-SS: An Object-Relationship-Attribute Model for Semi-Strucured Data. *Technical Report, Department of Computer Science, National University of Singapore*, 2001.

[7] Embley, D. and Mok, W.Y., Developing XML Documents with guaranteed "good" properties, *In Preceedings of the 20th International Conference on Conceptual Modeling*, 2001, pp. 426-441.

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

[9] Gustas, R., A Look Behind Conceptual Modelling Constructs in Information System Analysis and Design, *International Journal of Information System Modelling and Design*, 1(1), 2010, pp. 79-108

[10] Halpin, T., Object Role Modelling: Principle and Benefit, *International Journal of Information System Modelling and Design*, 1(1), 2010, pp. 33-55.

[11] Kolahi, S., Dependency-preserving normalization of relational and XML data, J*ournal of Computer and system Sciences,* 2007, pp. 636-647.

[12] Lee, S.Y., Lee, M.L., Ling, T.W., and Kalinichenko, L.A., Designing Good Semi-strucutred Databases, 1999, pp. 131-135.

[13] Ling, T.W and Yan, L.L, NF-NR: A practical Normal Form for Nested Relations, *Journal of System Integration*,4, 1994, pp. 303-340

[14] Ling, T.W, A normal form for entity-relationship diagram, *Proceeding 4th International Conference on Entity Relationship Approach*, 1985, pp. 24-35

[15] Ling, T.W., Lee, M.L. and Dobbie, G. Semi structured Database Design, Springer 2005.

[16] Lv, T., Gu, N., Yan, P., Normal forms for XML documents, *Information and Software Technology,* 2004, pp. 839-846.

[17] Mani, M., Lee, D., and Muntaz, R.R., Semantic Data Modeling Using XML Schemas, *In Proceeding of 20th International Conference on Conceptual Modelling*. 2001

[18] Mok, W. (2002). A comparative Study of Various Normal forms. *IEEE Transanctin on Knowledge and Data Enginnering, Vol.14* , pp. 369-385.

[19] Mok, W.Y., Ng, Y.K and Embley, D.W., A normal for precisely characterizing redundancy in nested relation, *ACM Transaction Database System*, Vol. 12, no.1, 1996. pp. 77-106

[20] Ozsoyoglu, Z.M and Yuan, L., A new normal form for Nested Relations, *ACM Transaction on Database System*, Vol. 12, No 1, 1987, pp.111-136.

[21] Vincet, M. , Liu, J., Mohania, M.,On the equivalence between FDs in XML and FDs in relations. *Acta Infomatica* , 2007, pp. 230-247.

[22] Wang, J. and Topor, R., Removing XML Data Redundancies Using Functional and Equality-Generating Dependencies, *16th Australasian Database Conference*, 2005, pp. 65-74.

[23] Yu, C. and Jagadish, J.H., XML schema refinement through redundancy detection and normalization, *The VLDB Journa*l, 2008, pp. 203-223.

[24] Yuliana, O.Y. & Chittayasothorn, S. (2005). XML Schema Re-Engineering Using a Conceptual Schema Approach. *Internatinal Conference on Information Technology: Coding and Computing.* IEEE.

[25] Zainol, Z. and Wang, B., GN-DTD: Graphical Notation for Describing XML Docuements, *In Preceeding of 2nd International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA*, IEEE Computer Society, 2010, pp. 214-221.