

Available from: http://dx.doi.org/10.1007/11775300_43

Copyright © 2006 Springer-Verlag Berlin Heidelberg. The original publication is available at www.springer.com.

This is the author's version of the work. It is posted here with the permission of the publisher for your personal use. No further distribution is permitted. If your library has a subscription to these conference proceedings, you may also be able to access the published version via the library catalogue.

Accessed from Swinburne Research Bank: http://hdl.handle.net/1959.3/5806
Designing Quality XML Schemas from E-R Diagrams

Chengfei Liu and Jianxin Li

Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, VIC 3122, Australia
{cliu, jili}@ict.swin.edu.au

Abstract. XML has emerged as the standard for representing, exchanging and integrating data on the Web. To guarantee the quality of XML documents, the design of quality XML Schemas becomes essentially important. In this paper, we look into this problem by designing quality XML Schemas from given E-R diagrams. We first discuss several criteria in designing a good XML Schema. Following these criteria, transformation rules are then devised that take all constructs of an E-R diagram into account. Finally, a recursive algorithm is developed to transform an E-R diagram to a corresponding quality XML Schema.

1 Introduction

XML has emerged as the standard for representing, exchanging and integrating data on the Web. Given that the structure of XML documents is much more flexible than that of a relational database, the design of a quality XML document for an application is non-trivial. By a quality XML document, we mean that it reflects the semantics of the application accurately and can be accessed, updated and integrated efficiently. To guarantee the quality of XML documents, the design of quality XML schemas becomes essentially important.

We reckon that several criteria need to be followed in designing a quality schema. (1) information preservation - it is fundamental that the target XML Schema preserves structural and semantic information of the application entirely. (2) highly nested structure - nesting is important in XML documents because it allows navigation of the paths in the document tree structures to be processed efficiently. (3) no redundancy - there is no data redundancy in the XML documents that conform to the target XML schema, thus no inconsistency will be introduced while updating the XML documents. (4) consideration of dominant applications - the structure of XML document should be accommodated such that dominant applications can be guaranteed to be processed efficiently. (5) reversibility of design - the original design can be achieved from the target XML schema, which is fundamentally important to data integration.

Kleiner and Lipect [1] proposed a method for generating XML DTD [2] from E-R diagrams. The method preserved as much structural information from E-R diagrams as possible. However, due to the limitation of the DTD, only annotations were
used to represent some E-R constructs that have no counterparts in DTD. Many-to-
many relationships were translated into top-level elements only so nesting is not
maximised. Some advanced features in E-R model such as ISA and aggregation were
not considered in their work. Bird et al. [3] proposed an approach to design XML
Schemas from the Object Role Model (ORM) [4]. The approach considered the
dominant applications by analysing the weighting and anchoring of factor types.
However, nesting was not discussed in their work. Effort has been put for translating
relational database schemas to XML Schemas. An early work in transforming rela-
tional schema to XML schema is DB2XML [5]. DB2XML uses a simple algorithm to
map flat relational model to flat XML model in almost one-to-one manner. DTD is
used for the target XML schema. Based on a flat translation similar to DB2XML, Lee
et al. [6] presented two algorithms NeT and CoT. NeT derives nested structures from
flat relations by repeatedly applying the nest operator on tuples of each relation. The
resulting nested structures may be useless because the derivation is not at the type
level. CoT considers inclusion dependencies as constraints to generate a more intu-
tive XML Schema. XViews [7] constructs a graph based on primary key/foreign key
relationship and generates candidate views by choosing the node with either maxi-
mum in-degree or zero in-degree as the root element. The candidate XML views
generated maybe highly nested. DTD is also chosen for target XML schema. This
approach does not consider the preservation of integrity constraints. It also suffers
considerable level of data redundancy. Liu et al. [8] proposed an approach that en-
sures the transformed schema in XML Schema [9] is highly nested, redundancy free
and preserves all the integrity constraints. However, the dominant applications and
the reversibility of transformation were not discussed in their work. Bohannon et al.
[10] developed the notion of DTD schema embedding that preserves information by
ensuring both effective invertible mapping and efficient XML query translation. Lots
of work has been done on mapping from XML to relational databases for storage
purpose. Recently, Barbosa et al. [11] proposed a framework for information-
preserving XML-to-relational mapping. The framework is extensible and guarantees
the target relational schema is equivalent to the original XML Schema.

We aim at designing quality XML Schemas that follows all five criteria we dis-
cussed above. Similar to conventional database design, we use E-R model [12] for
conceptual modelling, so we assume that E-R diagrams are given when we design
XML schema. In this paper, we present our transformation rules and algorithms that
automatically generate quality XML Schemas from E-R diagrams by following all
five criteria. To preserve information, we choose XML Schema as the target schema
language instead of DTD because XML Schema provides far more powerful model-
ling features than DTD.

The rest of the paper is organised as follows. In Section 2, we briefly introduce
the E-R model and XML Schema. Following our design criteria, we design transforma-
tion rules that consider all the constructs in the E-R model in Section 3. In Section
4, we propose a recursive algorithm that generates a quality XML schema from a
given E-R diagram. Section 5 concludes the paper.
2 E-R Model and XML Schema

Before we discuss the mapping from an E-R diagram to its correspondent schema in XML Schema, we briefly review both the E-R model and XML Schema.

The E-R model employs three basic notions: entities (entity sets), relationships, and attributes. There are two types of entity sets: regular and weak. The existence of a weak entity depends on another entity (its parent entity). A relationship has two basic properties: cardinality (one-to-one, one-to-many, many-to-many) and participation (total and partial). Two or more participants may be involved in a relationship. The former is called binary while the latter is called n-ary. Sometimes, a relationship may have participants that belong to same entity set and play different roles. This relationship is called a recursive relationship. The relationship from a parent entity set to a weak entity set is called identifying relationship. An attribute can be atomic or composite by having its own attributes, and meanwhile can be single-valued or multi-valued.

The set of attributes that can uniquely identify an entity in a regular entity set is called a key. The set of attributes that can identify a weak entity in the context of its parent entity is called a local key. A global key of a weak entity consists of its local key and the key of its parent entity. A key for a relationship consists of all keys of its participant entity sets.

The E-R model is also extended to support some advanced features. These include ISA (generalisation and specialisation), and aggregation where some relationships are treated as high-level entity sets.

To incorporate all the constructs introduced above, we give a formal definition in connection to an E-R diagram as follows.

Definition 1: An E-R diagram is represented $\delta = (E, R, A, \rho, n_d, s, p, k)$, where

1. $E$ is the set of entity sets. Each $e \in E$ is defined as $(n_e, t)$ where $n_e$ and $t$ are the name and type of $e$, and $t \in \{\text{regular, weak, high-level}\}$. If $t(e) = \text{"high-level"}$, $e$ has its own E-R diagram $\delta_e$ which includes a single relationship.

2. $R$ is the set of relationships. Each $r \in R$ is defined as $(n_r, \{(e, \text{card}, \text{par}, \text{role})\})$ where $n_r$ is the name of the relationship and each tuple $(e, \text{card}, \text{par}, \text{role})$ in the set is used to describe a participant entity set. The participant entity set, its cardinality, participation and role in the relationship are recorded. Here, $e \in E$, card $\in \{1, n\}$, par $\in \{\text{total, partial}\}$.

3. $A$ is the set of attributes. Each $a \in A$ is defined as $(n_a, \text{vt}, st)$ where $n_a$ is the attribute name, $\text{vt} \in \{\text{single-valued, multi-valued}\}$, and $st \in \{\text{atomic, composite}\}$.

4. $\rho : E \cup R \cup A \rightarrow 2^d$ defines the attribute sets of entities, relationships, and composite attributes.

5. $n_d$ is the name of the diagram.

6. $s : E \rightarrow E$ defines the ISA relationship. For $e \in E$, $s(e)$ is the super entity set of $e$.

7. $p : E \rightarrow E$ defines the identifying relationship. For a weak entity set $e \in E$, $p(e)$ is the parent entity set of $e$. 
3 Transformation Rules

To map all constructs of an E-R diagram defined in Section 2 and follow all criteria discussed in Section 1, we design the following set of transformation rules.

**Rule 1: E-R diagram** - For an E-R diagram \( \delta \) \((\mathcal{E}, \mathcal{R}, \mathcal{A}, \mathcal{P}, \mathcal{S}, \mathcal{K})\), a root element named \( \text{nd} \) is created as follows.

\[
\begin{aligned}
&<\text{xsd:element name="nd"}>
&\begin{aligned}
&<\text{xsd:complexType}>
&\begin{aligned}
&<\text{xsd:sequence}>
&\text{<!-- detail of transformed XML schema goes here -->}
&\end{aligned}
&\end{aligned}
&\end{aligned}
&\text{xsd:complexType}>
&\text{xsd:element}>
\end{aligned}
\]

**Rule 2: Regular entity set** - For an entity set \( e(n_e, t) \) of the E-R diagram \( \delta \) where \( t(e) = \text{"regular"} \), an element named \( n_e \) is created and put under the element for \( \delta \).

The key of \( e \) is specified by a key declaration where \( \mathcal{K}(e) = \{ k_1, \ldots, k_n \} \).

\[
\begin{aligned}
&<\text{xsd:element name="n_e"}>
&\begin{aligned}
&<\text{xsd:complexType}>
&\begin{aligned}
&<\text{xsd:sequence}>
&\text{<!-- detail of the entity set goes here -->}
&\end{aligned}
&\end{aligned}
&\end{aligned}
&\text{xsd:complexType}>
&\text{xsd:element}>
\end{aligned}
\]
Rule 3: Weak entity set - For an entity set \( e(n_e, t) \) of the E-R diagram \( \delta \) where \( t(e) = \text{“weak”} \) and \( \bar{t}(e) = e' \), an element named \( n_e \) is created and put under the element for \( e' \). The key of \( e \) is specified by a key declaration where \( k(e') = \{k_{11}, \ldots, k_{1m}\} \), \( k(e) = \{k_{21}, \ldots, k_{2n}\} \).

Rule 4: High-level entity set - For an entity set \( e(n_e, t) \) of the E-R diagram \( \delta \) where \( t(e) = \text{“high-level”} \), an element named \( n_e \) is created and put under the element for \( \delta \). A high-level entity set is used to represent one and only one relationship. As such, the key of \( e \) is the key of the relationship which can be achieved while generating the detail of the entity by applying Rule 1 to its own E-R diagram \( \delta_e \).

Dominant queries are those queries that are most frequently used. Instead of defining a dominant query, we define the dominant entity set (or role) of a relationship and the dominant relationship of an entity as follows.

Definition 2: Dominant entity set (or role): The dominant entity set \( e \) or role \( l \) of a relationship \( r(n_r, \{(e, \text{card}, \text{par}, \text{role})\}) \) is one of its participant entity sets or roles such that \( e \) or \( l \) has the highest frequency from which \( r \) is visited.

Definition 3: Dominant relationship: The dominant relationship \( r \) of an entity set \( e(n_e, t) \) is one of its participating relationships such that \( r \) has the highest frequency from which \( e \) is visited.

Rule 5: One-to-one relationship - For a relationship of the form \( r(n_r, \{(e_1, 1, p_1, \_), (e_2, n, p_2, \_))\} \), an element named \( n_r \) is first created, then depending on \( p_1 \) and \( p_2 \), apply different rules as follows.

1. both are \text{“total”} - suppose that \( e_1 \) is the dominant entity set, put the element for \( r \) under the element for \( e_2 \) and change to put the element for \( e_2 \) to under the element for \( e_1 \).
2. one of them, say \( p_1 \), is \text{“partial”} - put the element for \( r \) under the element for \( e_2 \) and change to put the element for \( e_2 \) to under the element for \( e_1 \).
3. both are \text{“partial”} - suppose that \( e_1 \) is the dominant entity set, put the element for \( r \) under the element for \( e_1 \). Foreign key attributes are added in \( r \) with a separate keyref declaration where \( k(e_2) = \{k_{11}, \ldots, k_{1n}\} \).

Rule 6: One-to-many relationship - For a relationship of the form \( r(n_r, \{(e_1, 1, p_1, \_), (e_2, n, p_2, \_))\} \), an element named \( n_r \) is first created, then depending on \( p_2 \), apply different rules as follows.
(1) \( p_1 \) is “total” - put the element for \( r \) under the element for \( e_2 \), then change the element for \( e_2 \) by adding \text{maxOccurs} = “unbounded” and move it to under the element for \( e_1 \).

(2) \( p_2 \) is “partial” - put the element for \( r \) under the element for \( e_2 \). Foreign key attributes are added in \( r \) with a separate keyref declaration where \( k(e) = \{ k_1, \ldots, k_n \} \).

\[
\text{Rule 7: Many-to-many relationship} - \text{For a relationship of the form } r(n_\text{r}, \{(e_1, n, _), (e_2, n, _)\}), \text{ an element named } n_\text{e} \text{ is first created with maxOccurs attribute set to “unbounded”, then put the element for } r \text{ under the element for the dominant entity set, say } e_1. \text{ Foreign key attributes are added in } r \text{ with a separate keyref declaration where } k(e_2) = \{ k_1, \ldots, k_n \}.\n\]

\[
<\text{xsd:keyref name="foreignKey}_r \text{ e}_2\text{" refer="key}_n\text{e}_1\text{"}>
<\text{xsd:selector xpath="path}_r\text{"} /><\text{xsd:field xpath="k}_1\text{"} /> \ldots <\text{xsd:field xpath="k}_n\text{"} />\n</\text:xsd:keyref>\n\]

\[
\text{Rule 8: Recursive relationship} - \text{For a relationship of the form } r(n_\text{r}, \{(e_1, c_1, _), (e_1, c_2, _), \ldots \}), \text{ depending on } c_1 \text{ and } c_2, \text{ apply different rules as follows.}
\]

1. \( c_1 = c_2 = “1” \) - suppose that \( r \) is the dominant role, an element named \( n_\text{e} r_1 \) is created and put under the element for \( e_1 \), and foreign key attributes for \( r \) are added with a separate keyref declaration.

2. \( c_1 = c_2 = “n” \) - suppose that \( r \) is the dominant role, an element named \( n_\text{e} r_1 \) is created with maxOccurs set to “unbounded” and put under the element for \( e_1 \), foreign key attributes for \( r \) are added with a separate keyref declaration.

3. \( c_1 \neq c_2 \) (suppose \( c_1 > c_2 \)) - an element named \( n_\text{e} r_1 \) is created and put under the element for \( e_1 \), and foreign key attributes for \( r \) are added with a separate keyref declaration.

\[
\text{Rule 9: ISA relationship} - \text{For an entity set } e_1, \text{ if } e_2 = \text{s}(e_1) \text{ is defined and the complexType defined for the element for } e_2 \text{ is } t_\text{e}_2, \text{ then an element for } e_1 \text{ can be created with the } t_\text{e}_2 \text{ as the extension type.}
\]

\[
<\text{xsd:element name="e}_1\text{"}>
<\text{xsd:complexType>
<\text{xsd:extension base="t}_e}_2\text{"}>
<\text{xsd:sequence>
<!– transformation of extra attributes of } e_1 \text{ goes here –>}
</\text:xsd:sequence>
</\text:xsd:extension>
</\text:xsd:complexType>
</\text:xsd:element>\n\]

\[
\text{Rule 10: N-ary relationship} - \text{For a relationship of the form } r(n_\text{r}, \{(e_1, c_1, _), \ldots, (e_n, c_n, _)\})), \text{ an element named } n_\text{e} \text{ is created and put under the element for the dominant entity set, say } e_1. \text{ Foreign key attributes are added in } r \text{ with } n-1 \text{ separate keyref declarations where } k(e_2) = \{ k_{21}, \ldots, k_{2m_1} \}, \ldots, k(e_n) = \{ k_{n1}, \ldots, k_{nn} \}. \text{ If exists } c_i = “n”, (2 \leq i \leq n), \text{ maxOccurs="unbounded" is added to the element.}
\]

\[
<\text{xsd:keyref name="foreignKey}_r \text{ e}_2\text{" refer="key}_n\text{e}_2\text{"}>
<\text{xsd:selector xpath="path}_r\text{"} /><\text:xsd:field xpath="k}_{21}\text{"} /> \ldots <\text:xsd:field xpath="k}_{2m_1}\text{"} />\n</\text:xsd:keyref>\n\]
Rule 11: Composite attribute - For an attribute \(a(n_a, \nu, s)\) where \(st(a) = \text{"composite"}\), of the entity set \(e\) or relationship \(r\) or composite attribute \(a'\), an element named \(n_a\) is created and put under the element for \(e\) or \(r\) or \(a'\). maxOccurs="unbounded" is added to the element if \(\nu(a) = \text{"multi-valued"}\).

Rule 12: Atomic attribute - For an attribute \(a(n_a, \nu, s)\) where \(st(a) = \text{"atomic"}\), of the entity set \(e\) or relationship \(r\) or composite attribute \(a'\), different rules apply depending on \(\nu(a)\).

(1) If \(\nu(a) = \text{"multi-valued"}\), an element named \(n_a\) is created and put under the element for \(e\) or \(r\) or \(a'\). maxOccurs="unbounded" is added to the element. The type of \(a\) is specified in the type attribute of the element.

(2) If \(\nu(a) = \text{"single-valued"}\), either an attribute named \(n_a\) associated with the element for \(e\) or \(r\) or \(a'\), or an element named \(n_a\) can be created and put under the element for \(e\) or \(r\) or \(a'\).

From the above transformation rules, it is easy to find that
- Information preservation and design reversibility criteria have been considered in all the transformation rules.
- Highly nested structure criterion has been taken into account in Rules 3, 5, 6 and 7.
- No redundancy criterion has been applied in Rules 3 and 6.
- Dominant applications criterion has been used in Rules 5, 7 and 8.

4 Mapping E-R Diagrams to XML Schemas

Given an E-R diagram \(\delta (E, R, A, \rho, \eta, \sigma, \pi, k)\), we design a transformation algorithm called ERD2XSD to generate a corresponding XML schema by applying the transformation rules introduced in the previous section. Normally an ISA relationship only applies to regular entity sets. In ERD2XSD, we first generate XML schema elements for regular entity sets (Line 4-9) and ISA relationships (Line 10-14), then generate elements for all other entity sets (Line 15-26). If an entity set is of type ‘high-level’, the algorithm is called recursively to transform the E-R diagram of the high-level entity set first. The weak entity sets are processed after the regular and high-level entity sets because of the global key derivation caused by the existence dependency. If a weak entity set \(e_i\) depends on another weak entity set \(e_j\), \(e_i\) will also be processed after \(e_j\). After that, relationships are processed (Line 27-49). The order for transforming relationships is considered carefully in the algorithm such that nesting of one entity set under another is done just once. Finally, XML schema elements/attributes are generated for composite or atomic attributes in the diagram \(\delta\) (Line 50-54).
4.1 Transformation Algorithm

The algorithm \textit{ERD2XSD} is given below.

\textbf{Algorithm : \textit{ERD2XSD}}

\textbf{Input}: an E-R diagram $\delta$ $(E, R, A, \rho, n_d, s, p, k)$

\textbf{Steps}:
1. apply Rule 1 to create the root element for $\delta$;
2. $E_1 = \{e | e \in E \land t(e) = \text{"regular"}\}$;
3. $E_2 = E - E_1$;
4. for each $e \in E_1$ { /* process "regular" entity sets without supersets */
   5. if $s(e)$ is not defined {
      6. apply Rule 2 to generate the element for $e$;
      7. $E_1 = E_1 - \{e\}$;
   8. }
9. }
10. while $E_1 \neq \emptyset$ do { /* process ISA relationships */
   11. get $e \in E_1$ such that $s(e) \notin E_1$; /* no dependency on entity sets in $E_1$ */
   12. apply Rule 9 to generate the element for $e$ based on its $s(e)$;
   13. $E_1 = E_1 - \{e\}$;
   14. }
15. while $E_2 \neq \emptyset$ do { /* process "high-level" and "weak" entity sets */
   16. get $e \in E_2$;
   17. if $t(e) = \text{"high-level"}$ {
      18. apply Rule 4 to generate the element for $e$;
      19. \textit{ERD2XSD}($\delta_e$); /* recursively processing $e$ */
      20. $E_2 = E_2 - \{e\}$;
   21. }
22. else if $\rho(e) \in E_2$ { /* check dependency between weak entity sets */
   23. apply Rule 3 to generate the element for $e$;
   24. $E_2 = E_2 - \{e\}$;
   25. }
26. }
27. $R_1 = R$; /* process relationships other than 1:1 relationships */
28. for each $r \in R_1$ {
   29. if nary($r$) > 2 { /* nary($r$) returns the number of participant entity sets */
      30. apply Rule 10 to generate and nest the element for $r$;
      31. $R_1 = R_1 - \{r\}$;
   32. }
33. else if nary($r$) = 1 {
   34. apply Rule 8 to generate and nest the element for $r$;
   35. $R_1 = R_1 - \{r\}$;
   36. }
37. else if card($r.e1$)="n" \& card($r.e2$)="n" { /* card($r.e$) returns cardinality of $e$ in $r$ */
   38. apply Rule 7 to generate and nest the element for $r$;}
39. \[ R1 = R1 - \{r\}; \]
40. \} 
41. \textbf{else if} \ \text{card}(r.e1) = "n" \text{\lor} \text{card}(r.e2) = "n" \} 
42. \textbf{apply} Rule 6 to generate and nest the element for \( r \) and to adjust the nesting of \( e1 \) and \( e2 \); 
43. \[ R1 = R1 - \{r\}; \]
44. \} 
45. \} 
46. \textbf{for each} \( r \in R1 \) \{ /* the remaining relationships are all 1:1 */ 
47. \textbf{apply} Rule 5 to generate and nest the element for \( r \); 
48. \[ R1 = R1 - \{r\}; \]
49. \} 
50. \textbf{for each} \( a \in A \) \{ /* the remaining relationships are all 1:1 */ 
51. \textbf{if} \ \text{st}(a) = "atomic" \textbf{apply} Rule 12 to generate and nest an element/attribute for \( a \); 
52. \textbf{else apply} Rule 11 to generate and nest an element for \( a \); 
53. \[ A = A - \{a\}; \]
54. \} 

\textbf{Output}: the root element named \( nd \) for \( \delta \) in the target XML schema

\section*{4.2 Transformation Example}

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure1.png}
\caption{An example E-R diagram \textit{company}}
\end{figure}

Figure 1 shows an E-R diagram named \textit{company}. The keys and local keys for regular and weak entity sets are underlined with solid and dotted lines, respectively. Given this E-R diagram as input to the ERD2XSD algorithm, the XML schema with the following root element will be generated.

\begin{verbatim}
<xsd:element name="company">
  <xsd:complexType>
    ...
  </xsd:complexType>
</xsd:element>
\end{verbatim}
<xsd:sequence>
  <xsd:element name="employee" type="t_employee" maxOccurs="unbounded"/>
  <xsd:element name="job" maxOccurs="unbounded"/>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="worksOn" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="projID" type="xsd:int"/>
          <xsd:attribute name="ssn" type="xsd:int"/>
          <xsd:attribute name="workTime" type="xsd:time"/>
        </xsd:complexType>
        <xsd:keyref name="foreignKey_worksOn_employee" refer="key_employee">
          <xsd:selector xpath="/worksOn"/>
          <xsd:field xpath="@ssn"/>
        </xsd:keyref>
        <xsd:keyref name="foreignKey_worksOn_project" refer="key_project">
          <xsd:selector xpath="/worksOn"/>
          <xsd:field xpath="@projID"/>
        </xsd:keyref>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="jobID" type="xsd:int" use="required"/>
    <xsd:attribute name="hourlyRate" type="xsd:int"/>
    <xsd:key name="key_job">
      <xsd:selector xpath="/job"/>
      <xsd:field xpath="@jobID"/>
    </xsd:key>
  </xsd:complexType>
  <xsd:element name="department" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="clerk" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:extension base="t_employee">
              <xsd:sequence>
                <xsd:element name="worksFor"> … … </xsd:element>
              </xsd:sequence>
            </xsd:extension>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="manager">
          <xsd:complexType>
            <xsd:extension base="t_employee">
              <xsd:sequence>
                <xsd:element name="manage">
                  <xsd:complexType>
                    <xsd:sequence>
                      <xsd:element name="startDate" type="xsd:date"/>
                    </xsd:sequence>
                  </xsd:complexType>
                </xsd:element>
              </xsd:sequence>
            </xsd:extension>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:sequence>
<xsd:complexType>
  <xsd:element name="project" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="control">
          <xsd:complexType>
            <xsd:attribute name="duration" type="xsd:duration"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="projID" type="xsd:int" use="required"/>
      <xsd:attribute name="name" type="xsd:string"/>
      <xsd:key name="key_project">
        <xsd:selector xpath="/project"/><xsd:field xpath="@projID"/>
      </xsd:key>
    </xsd:complexType>
  </xsd:element>
</xsd:complexType>

<xsd:complexType>
  <xsd:element name="department">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:attribute name="deptID" type="xsd:int" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:key name="key_department">
          <xsd:selector xpath="/department"/><xsd:field xpath="@deptID"/>
        </xsd:key>
      </xsd:sequence>
      <xsd:attribute name="ssn" type="xsd:int"/>
      <xsd:attribute name="name" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:complexType>

<xsd:complexType name="t_employee">
  <xsd:sequence>
    <xsd:element name="supervise_emp">
      <xsd:complexType>
        <xsd:attribute name="mgr" type="xsd:int"/>
        <xsd:keyref name="foreignKey_supervise" refer="key_employee">
          <xsd:selector xpath="/supervise_emp"/><xsd:field xpath="@mgr"/>
        </xsd:keyref>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="dependent" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="ssn" type="xsd:int"/>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
5 Conclusion

In this paper, we first discussed design criteria of a good quality XML schema. Then, we designed transformation rules that translate all the constructs of an E-R model into their counterparts in XML Schema. We claimed that these set of rules follow all five criteria discussed in the paper, i.e., information preservation, highly nested structures, no redundancy, consideration of dominant applications, and design reversibility. Based on the transformation rules, a recursive algorithm called ERD2XSD was proposed that takes an arbitrary E-R diagram as input and generates a correspondent high quality XML schema as output. An illustrative example was given in the end. In the future, we will build a prototype using this algorithm and improve it as a real XML schema design tool.

Acknowledgement

This work was supported by the Australian Research Council Discovery Project under the grant number DP05559202.

Reference