An algorithm for constructing XML Schema documents from relational databases

Abdelsalam M. Maatuk
Faculty of Information Technology
Benghazi University, Libya
abdelsalam.maatuk@uob.edu.ly

M. Akhtar Ali
Faculty of Engineering and Environment
Northumbria University, UK
akhtar.ali@northumbria.ac.uk

Shadi Aljawarneh
Software Engineering Department
Jordan University of Science and Technology, Irbid, Jordan

ABSTRACT
The aim of this paper is to present a solution to automatically generate an XML schema from an existing relational database (RDB). The important goal of this translation is to enriching the source schema using semantics that might have not been clearly expressed in it, by acquiring as much information as possible about objects and relationships that exist among them. The next step is to produce an enhanced meta data model, which captures essential characteristics of target XML schema, and is suitable for translation. In details, we present translations of all constructs of an RDB into an XML Schema and integrated these into an algorithm. This process is to simplify exchange of data between different databases, practically the import of data of RDBs into XML documents. A prototype has been developed to realize the algorithm and generate target schema. To validate our proposal, we present experimental results using both schemas. The results show that the proposed algorithm is correct.

Keywords
Re-engineering databases; XML data conversion; semantic enrichment; schema translation.

1. INTRODUCTION
The increasing popularity of non-traditional applications (e.g., multimedia, geographical information systems) can be considered to be among the most significant recent changes in information technology. These novel technologies have been dominant in the area of information systems due to their productivity and flexibility. In addition, the emergence of e-technology applications have led to the development of languages and tools for exchanging and publishing relational database (RDB) data over the Web. However, as the majority of databases are still maintained in relational database management systems, therefore, it is expected that the need to convert or publish such RDBs into the technologies that have emerged recently will grow substantially [4].

XML which support diverse concepts, have been proposed in order to fulfill the demands of complex applications that require rich data types. XML is nowadays used as a database at content level and as a dominant standard at hypertext level [5]. XML is a powerful model because it extends simple user-defined tags to more levels with complex structures and relationships such as aggregation and inheritance. XML Schema language is a standard that provides a sophisticated means for describing the structures and constraints of schema and instance documents [14]. Moreover, it borrows concepts from RDB models such as key and integrity constraints, and other concepts from object-based models such as inheritance, references, data collections and user-defined data types. Most existing methods for translating RDB schema into XML focus on generating a document type definition (DTD). As the XML Schema standard has gained a wide acceptance in recent years for more independence, it is important that database conversion methods generate target databases according to this standard.

In this paper, we present a solution to construct an XML schema document from an existing RDB. The first step in this translation is to enrich a source RDB semantically by acquiring as much metadata information as possible, and produce an enhanced metadata model called Canonical Data Model (CDM), which captures essential characteristics of target XML schema. The CDM then guides the translation of RDB schema into the target XML Schema, ensuring that the conversion process is accomplished with data integrity and consistency. We present a set of translation rules, integrated into an algorithm, to translate all constructs of an RDB into an XML Schema. We have developed a prototype to realize the algorithm and proof of the concept. We conducted an experiment to validate our approach using real databases. The experimental results demonstrate that the proposed algorithm is correct. Our method is more efficient in contrast to the existing solutions as it produces XML schema and exploits the powerful features provided by the XML Schema standard. The generated XML schema might help in the heterogeneous systems, publishing and sharing business data.

The paper is organized as follows. In Section 2 we present an overview of related works. Semantic enrichment of RDB in the form of CDM is described in Section 3. Section 4 shows how to translate the CDM into XML Schema. Section 5 explains the evaluation of the approach and the experimental results. Section 6 concludes the paper.

2. RELATED WORK
There are two approaches related to converting an RDB to XML. The first approach is for handling data stored in RDBs through XML interfaces so that it deals with schema translation, whereas the second approach is to migrate an RDB into XML database where both schema and data are completely migrated into a target database. Existing work on converting RDBs into XML documents enforces different prerequisites and made certain assumptions to facilitate the conversion process, which might be a

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ICEMIS '15, September 24-26, 2015, Istanbul, Turkey © 2015 ACM.
ISBN 978-1-4503-3418-1/15/09 $15.00
DOI: http://dx.doi.org/10.1145/2832987.2833007

point of limitations or a drawback. For example, a few work uses data dictionaries and well-designed RDB assumed[4] whereas other solutions consider legacy RDB for translation into XML documents [13]. Besides, the resulting XML schemas might be a DTD [4], XML Schema [13] or other independent XML language [3]. However, several researchers have proposed methods for mapping UML class diagrams to XML [12]. We have given a detailed review of translation solutions for various directions, which are related to database conversion in [15].

A method in which data semantics are extracted from an RDB schema into an EER model, which is mapped into an XSD graph is introduced in [6]. However, the authors suggested mapping foreign keys into a hierarchy of elements, which may cause redundancy when an element has a relationship with more than one element. Wang et al. [13] proposed a method for generating XML document from legacy RDBs using an ER model. However, inheritance and aggregation relationships are not considered. Lee et al. [4] presented an algorithm that maps RDBs into DTDs. However, the algorithm neither utilizes features provided by the XML model nor considers integrity constraints. RDB data can be published as XML documents, using special declarative languages, to be exchanged over the Web. SEML [7], XPERANTO [1] and XTABLES [8] are among the systems taking this approach. Through converting an RDB into XML, users see views that can be queried using XML query languages. However, data in such applications is not fully materialized in XML form, whereas the results are.

It could be concluded that research into the translation of RDB schemas into XML is still immature, and that therefore several areas are in need of further attention. We have noted that most works for converting to XML have used source-to-conceptual-to-target techniques, focusing on a DTD schema. Some semantics (e.g., inheritance, aggregation) are not considered in some work. This is mainly due to their lack of support for such semantics either in source or target data models, e.g., ER model and DTD lack support for inheritance. Compared to DTD, the XML Schema offers a much more extensive data types, and provides referencing and inheritance mechanisms of attributes and elements.

3. RELATIONAL DATABASE ENRICHMENT

The semantic enrichment process starts by extracting the metadata information about an RDB, including relation names and attribute properties (e.g., attribute names, data types, and whether the attribute is nullable), and Primary Keys (PKs), Foreign Keys (FKs) and Unique Keys (UKs). For this task, we have applied our approach [9] for enriching RDBs. We assume that data dependencies are represented by PKs and FKs as for each FK value there is an existing, matched PK value. The next step is to identify the CDM constructs based on a classification of relations, attributes and relationships. Lastly, the CDM structure is generated.

Definition 1: The CDM is defined as a set of classes:
$CDM := \{C \mid C := \langle cn, cls, abs, A_{cdm}, Rel, UK\rangle\}$, where each class *C* has a name *cn*, is given a classification *cls*, and whether or not it is abstract *abs*. Each *C* has a set of attributes A_{cdm}, a set of relationships *Rel*, and a set of unique keys *UK*.

Classification (*cls*): our classification scheme divides classes into the following categories:

1. Regular Strong Class (RST): a class whose PK is not composed of any FKs.
2. Secondary Strong Class (SST): an inherited RST.
3. Sub-class (SUB): a class that inherits another super-class, but is not inherited by other sub-classes.
4. Secondary Sub-class (SSC): a sub-class that is inherited by other sub-classes.
5. Secondary Relationship Class (SRC): a referenced RRC class, an M:N relationship class with attributes, or n-ary relationships, where n>2.
6. Regular Component Class (RCC): a weak class that participates in a relationship with other classes rather than its parent class.
7. Multi-valued Attribute Class (MAC): a class that represents a multi-valued attribute.
8. Composite Attribute Class (CAC): a class that represents a composite attribute.
9. Regular Relationship Class (RRC): an M:N relationship class without attributes.

Abstraction (*abs*): A super-class is *abstract* (i.e., *abs* :=**true**) when all of its objects are members of its sub-type objects. Instances of an abstract type cannot appear in the database extension, but are subsumed into instances of its sub-types.

Attributes (A_{cdm}): A class *C* has a set of attributes A_{cdm}.
$A_{cdm} := \{a \mid a := \langle a_n, t, tag, l, n, d\rangle\}$, where each attribute *a* has a name a_n, data type *t* and a *tag*, which classifies *a* as a non-key 'NK', 'PK', 'FK' or both PK and FK 'PF' attribute. Each *a* can have a length *l* and may have a default value *d*, whereas *n* indicates whether or not *a* is nullable ('y' | 'n').

Relationships (*Rel*): A class *C* has a set of relationships *Rel*. Each relationship *rel*∈*Rel* between *C* and class *C'* is defined in *C* to represent an association, aggregation or inheritance. $Rel := \{rel \mid rel := \langle RelType, dirC, dirAs, c, invAs\rangle\}$, where *RelType* is a relationship type, *dirC* is the name of *C'*, and *dirAs* denotes a set containing the attribute names representing the relationship from the *C'* side. The *invAs* denotes a set of inverse attribute names representing the inverse relationship from the *C* side, and *c* is the cardinality constraint of *rel* from the *C* side. *RelType* can have the following values: *'associated with'* for association, *'aggregates'* for aggregation, and *'inherits'* or *'inherited by'* for inheritance. Relationships have two cases: 1:1 and 1:M, and *c* is defined by *min..max* notation to indicate the minimum and maximum occurrences of objects of *C'* within objects of *C*. Based on *c*, the object(s) of *C'* can be single-valued where *c* := 0..1 (optional) or *c* := 1..1 (required), or set-valued where *c* := 0..* (optional) or *c* := 1..* (required).

Unique keys (*UK*): A class *C* may have a set of UK(s) that are preserved in *UK*: $UK := \{\delta \mid \delta := \{\langle ua, s\rangle\}\}$, where δ represents one key, *ua* is an attribute name, and *s* is a sequence number.

3.1 Constructing CDM from RDB

Using key matching, relations and their attributes are classified, relationships among relations are identified and their cardinalities are determined. All these are translated into equivalents in the CDM. The semantically enriched CDM can be then translated into the target schema. Each relation *R* is classified based on the comparison of its PK with the PKs of other relations, and mapped into one of the nine CDM classes above. After class *C* is classified, it is important, if *C.cls* := ("SST" | "SSC"), to check

whether *C* is concrete or abstract. *C* is a concrete class (i.e., *abs* := **false**) when some of its corresponding RDB table rows are not members of other sub-tables, and abstract otherwise. Attributes of *R* are identified and mapped into attributes of *C*. The keys of *R* are used to generate the relationships *Rel* of *C*. Using this information, the relationships are identified, their cardinalities determined, and they are mapped into *Rel* as association, inheritance or aggregation. Using corresponding data, every relationship that *R* participates in is identified and mapped into an equivalent relationship *rel* and added to *Rel*.

Example 1: Consider the RDB shown in Figure 1. PKs are in **bold** and FKs are in *italics*. Table 1 shows (partly) the resulting CDM. Each RDB relation is mapped into a class in CDM. For instance, the relation **Emp** is mapped into the CDM class **Emp**. The **Emp** class, which is an abstract SST class, has the attributes: *ename*, *eno*, *bdate*, *address*, *spreno* and *dno*. Other properties of the attributes (e.g., types, tags, length) are also shown. The **Emp** class is '*associated with*' the classes: **Dept** (twice), **Works** on and with itself (twice). Moreover, it '*aggregates*' the **Kids** class and is '*inherited by*' the **Salaried_emp** and **Hourly_emp** classes. Cardinality *c* and unique keys are also given for each class.

Emp (**eno**, ename, bdate, address, *spreno*, *dno*):
spreno → **Emp**, *dno* → **Dept**
Kids (***eno*, kname**, sex): *eno* → **Emp**
Salaried_emp (***eno***, salary): *eno* → **Emp**
Hourly_emp (***eno***, pay_scale): *eno* → **Emp**
Dept (**dno**, dname, *mgr*, startd): *mgr* → **Emp**
Dept_locations (***dno*, location**): *dno* → **Dept**
Proj (**pnum**, pname, plocation, *dnum*): *dnum* → **Dept**
Works_on (*eno, pno*): *eno* → **Emp**,*pno* → **Proj**

Figure. 1 Sample input RBD

4. TRANSLATING CDM INTO XML SCHEMA

This section explains how to translate CDM into an XML Schema file (.**xsd**). We first define the XML target schema, and then explain the steps of the algorithm for translating the CDM constructs into their equivalents in the target schema.

4.1 XML Schema standard definition

The structure of an XML document is made up of essential components such as annotations, element declarations and type definitions. Moreover, the document may contain other components such as attribute and model groups [11]. Besides, the XML Schema language [14] standard provides declaration of identity constraints, by which relationships and integrity constraints can be defined. PKs, FKs and UKs can be defined within the schema's root element using the **key**, **refkey** and **unique** elements, respectively. There are several mechanisms in the XML Schema that handle inheritance relationships such as derived types, substitution groups and abstract type mechanisms. Besides, a super-class complex type can be declared as **abstract** if all its instances are inherited by instances of its sub-classes. The multiplicity of elements is specified by **minOccurs** and **maxOccurs**. From this, the potential target XML Schema can be generated as two components. One is a global element, which represents the root of the XML tree defined as a complex type, containing schema elements and constraints. The second is a set, containing all global complex types. Each type can be used as a type of one element (or more) declared in the root or in other complex types. An inheritance is represented using the **complexContent**, **extension** and **base** keywords.

Definition 2: A target XML Schema is denoted as a 2-tuple: ‹*Root*, *GT*›, where *Root* is a global element declared under the schema with its direct local elements and constraints, and *GT* is a set consisting of global complex types. *GT* contains types of the elements declared in *Root*, or to be referenced by other types in *GT*. *Root* and *GT* are defined as follows:

- *Root* := ‹$root_n$, *LE*, PK_x, FK_x, UK_x›, where *Root* has a name $root_n$, a set of elements *LE*, and three sets of identity-constraints PK_x, FK_x and UK_x.
  - *LE* represents the complex type of *Root* that involves a set of local sub-element declarations:
    LE := {*e* | *e* := ‹e_n, e_t, *nim*, *max*›}, where each element *e* has a name e_n, a type e_t, and a minimum *min* and maximum *max* occurrences. The e_t is defined globally under the schema.
  - $PK_x$is a set of primary keys for the elements defined in *Root*, where:
    PK_x:= {*pk* | *pk* := ‹pk_n, *selector*, *PKfield*›}. Each primary key *pk* has a name pk_n, an element set *selector* as a scope within which the key is defined, and a set of sub-elements *PKfield* selected to be unique.
  - $FK_x$is a set of foreign keys, where:
    FK_x:= {*fk* | *fk* := ‹fk_n, *ref*, *selector*, *FKfield*›}. Each foreign key *fk* has a name fk_n, an element set scope *selector*, a reference constraint name *ref* that points to a matched primary key name, and a set of related sub-elements *FKfield*.
  - $UK_x$is a set of unique keys, where:
    UK := {*uk* | *uk* := ‹uk_n, *selector*, *UKfield*›}. Each unique key *uk* has a name uk_n, an element set scope *selector*, and a set of related sub-elements *UKfield* selected to be unique.
- *GT* := {*compType* | *compType* := ‹ct_n, *base*, *abst*, *LE*›}, where ct_n is the name of a complex type *compType*, *base* is the name of its super-type (if it is derived from another type), *abst* denotes whether or not *compType* is abstract type, and *LE* is a set of elements that are declared locally within *compType*.*LE* := {*e* | *e* := ‹e_n, e_t, *nim*, *max*›},is defined as for *Root*; however, e_t can be a built-in data type (e.g., a string) or a complex type pre-defined in the set *GT*.

4.2 The XML schema translation algorithm

This subsection explains how the CDM is translated into an equivalent target XML schema using a set of translation rules, which are described as follows:

4.2.1 Defining XML namespaces

XML Schema documents have main components, e.g., complex types, and secondary components, e.g., namespaces, and annotations. The secondary components must be defined in the first step to create an XML schema. A namespace is defined according to the standard for schema commands and assigned to a variable, e.g., **xs** as an XML Schema description using the attribute **xmlns** namespace. All schema tags are prefixed by **xs**: to indicate the schema namespace.

4.2.2 Declaring Schema root and its elements

In this paper, the target XML Schema is produced according to the Venetian Blind design [11], which defines complex types globally and elements locally. This offers flexible component reusing and nest element declarations.

Table 1 Results of CDM Generation

cn	*cls*	*abs*		A_{cdm}					*REL*					*UK*	
			an	*t*	*tag*	*l*	*N*	*d*	*relType*	*dirC*	*dirAs*	*C*	*invAs*	*ua*	*s*
Emp	SST	**true**	eno	int	PK	25	N		*asso*	**Dept**	dno	1..1	dno		
			ename	char		40	n		*asso*	**Dept**	mgr	0..1	eno		
			bdate	date			y		*asso*	**Emp**	eno	1..1	spreno		
			address	char		40	y		*asso*	**Emp**	spreno	0..*	eno		
			spreno	int	FK	25	y		*asso*	**Works_on**	eno	1..*	eno		
			dno	int	FK		n		*aggr*	**Kids**	eno	0..*	eno		
									inherBy	**Salaried_emp**	eno	1..1	eno		
									inherBy	**Hourly_emp**	eno	1..1	eno		
Salaried_emp	SUB	**false**	eno	int	PF	25	n		*inherts*	**Emp**	eno	1..1	eno		
			salary	int			y								
Dept	RST	**false**	dno	int	PK		N		*asso*	**Emp**	eno	1..1	mgr	mgr	1
			dname	char		40	n		*asso*	**Emp**	dno	1..*	dno		
			mgr	int	FK	25	n		*asso*	**Proj**	dnum	1..*	dno		
			startd	date			y		*aggr*	**Dept_locations**	dno	1..*	dno		
Works_on	RRC	**false**	eno	int	PF	25	N		*asso*	**Emp**	eno	1..1	eno		
			pno	int	PF		n		*asso*	**Proj**	pnum	1..1	pno		
asso: associated with *aggr: aggregates* *inherBy : inherited by*															

After defining the annotations, the root *Root* of the schema document is created and given an appropriate name $root_n$. The subsequent steps of the algorithm define the set of elements of the root *Root.LE* and its identity constraints PK_x, FK_x and UK_x, and then specify the set of global complex types *GT*. The target XML Schema document is generated from *Root* and *GT*.

Each CDM main and concrete class $C_{cdm} \in cdm$ (i.e., $C_{cdm}.abs$:= **false** and$C_{cdm}.cls \neq$ ('MAC' | 'CAC' |'RRC')) is translated as an empty first-level element under the root - placed in *Root.LE*. Each element $\in$*Root.LE* is named with the same name as the corresponding CDM class, i.e., $C_{cdm}.cn$ and has a type specified by adding the '**_t**' string to its name, i.e., $C_{cdm}.cn$+'_t'. The type name is used as a reference to a global type that is defined separately in the set *GT*. The occurrence of each element is specified using the occurrence *mn* := "**0**" and *mx* := "**unbounded**".

4.2.3 Defining complex types

Each CDM class C_{cdm}, where $C_{cdm}.cls \neq$('MAC' | 'RRC') is translated into a global complex type *ct* (of type *compType*). The name $ct_n$of *ct* is specified from the corresponding CDM class name $C_{cdm}.cn$, concatenated with the string '**_t**'. If C_{cdm} is abstract, i.e., $C_{cdm}.abs$:= **true**, then *ct* is specified as abstract, i.e., *ct.abst* := $C_{cdm}.abs$. The set of elements *ct.LE* is constructed from $C_{cdm}.A_{cdm}$ and $C_{cdm}.REL$. Each attribute $a \in C_{cdm}.A_{cdm}$is translated into a local element and added to *ct.LE*. Each element is given a name as the same name of the corresponding *a*, and a type translated into an equivalent data type according the target schema. The occurrences *min* and *max* of the element, are set to default values where each is "**1**" since they are all of the primitive type. However, *min* is set to "**0**" if *a* accepts nulls (i.e., *a.n* := 'y'). Foreign key attributes are defined in *ct* as simple attributes if they are specified in PK_x and FK_x sets; otherwise, they are dropped from the definition of *ct* (e.g., foreign key attributes in CAC classes). In other words, each $a \in C_{cdm}.A_{cdm}$ are mapped into a local element, where $C_{cdm}.cls$:= ('CAC' | 'SUB' | 'SSC') and *a.tag* $\neq$ 'PF'; whereas all attributes of C_{cdm}, where $C_{cdm}.cls$:= ('RST' | 'RCC' | 'SRC') are mapped in *ct* into local elements.

4.2.4 Translating relationships and constraints

An XML Schema represents relationships among elements using two techniques:

(i) by specifying nested complex types, or
(ii) constraints using the **key**/**keyref**.

We follow each of these two techniques with the aim of producing less data redundancy in a nested document. Thus, relationships among main CDM classes are mapped into identity constraints using the **key**/**keyref**, whereas the MAC, CAC and RRC classes are translated as nested elements under their parent elements.

Identity Constraints: The sets PK_x, FK_x and FK_x are declared in *aRoot* from each corresponding CDM class C_{cdm} using $C_{cdm}.A_{cdm}$ and $C_{cdm}.REL$. The following functions are defined to return the three sets:

- *definePK* (C_{cdm}) returns the primary key *pk* for each element defined under *Root* in the form ‹pk_n, *selector*, *PKfield*›, and adds it into the PK_xset. The *pk* element is translated from each attribute $a \in C_{cdm}.A_{cdm}$, where $C_{cdm}.cls$:= ('RST' | 'SRC' | 'RCC') and *a.tag* := ('PK' | 'PF'). To guarantee the uniqueness, each key name $pk_n$is formed by concatenating $C_{cdm}.cn$ with each *a.an* and the string 'PK'. A *selector* is assigned $C_{cdm}.cn$ as a constraint element scope, whereas *PKfield* is specified from each $a \in C_{cdm}.A_{cdm}$when *a.tag* := ('PK' | 'PF') as related element to the selected *selector*. The *PKfield* can have more than one element in the case of a composite key. For example, the primary key *dno* of **Dept** class is translated into the XML primary key as ‹"**deptDnoPK**", **Dept**, {**dno**}›. However, if $C_{cdm}.cls$:= ('SUB' | 'SSC'), the set *PKfield* contains the key attributes of the top super-type of the inheritance hierarchy.
- *defineFKs*(C_{cdm}) returns foreign keys for each element defined under *aRoot* in the form ‹fk_n, *ref*, *selector*, *FKfield*› and adds them to the FK_xset.XML foreign keys are mapped from each CDM relationship *rel* $\in$ $C_{cdm}.REL$, where *rel.relType* := *'associated with'*, and the attribute names in *rel.invAs*, which are tagged as 'FK' or 'PF'. The foreign key name $fk_n$is formed by the concatenation of $C_{cdm}.cn$, with the name of each attribute in *rel.invAs*, and the string 'FK'. The *ref* is formed from concatenating the *rel.dirC*, with the attribute names in *rel.dirAs* and the string 'PK', whereas the *selector* is named as $C_{cdm}.cn$, and each element in *FKfield* is assigned an attribute in *rel.invAs*.
- *defineUKs*(C_{cdm}) returns the unique keys for elements defined under *aRoot* and adds them to the set UK_x, based on their equivalents in CDM, i.e., $C_{cdm}.UK$.

Nested Elements: The following rules are used to translate CDM relationships into XML as sub-elements embedded within their parent elements.

- Each *rel* ∈ C_{cdm}.*REL* between C_{cdm} and another CDM class C'_{cdm}, where C'_{cdm}.*cls* := 'RRC', is translated into a multi-valued element in a complex type *ct* of the element translated from C_{cdm}. As C'_{cdm} participates in only two M:1 associations with C_{cdm} and another CDM class C''_{cdm}, *rel* is mapped in *ct* into a multi-valued element, referencing a complex type *ct''*, corresponding to C''_{cdm}. A foreign key is defined for the new sub-element through its parent element and added to FK_x, referencing the primary key of the element corresponding to C''_{cdm}. Similarly, a multi-valued element with its foreign key is defined in *ct''*, referencing *ct*.
- There are two different mapping rules that can be applied when *rel* ∈ C_{cdm}.*REL* represents an aggregation relationship, i.e., where *rel.relType* := *'aggregates'*, between a parent class C_{cdm} and a component class C'_{cdm}. A sub-element corresponding to C'_{cdm} is defined and embedded in the parent complex type *ct*, mapped from C_{cdm}, representing this relationship. The sub-element occurrences *min* and *max* are declared according to the corresponding cardinality *rel.c*. If C'_{cdm}.*cls* := 'MAC', *rel* is mapped into a simple multi-valued sub-element. However, if C'_{cdm}.*cls* := 'CAC', *rel* is mapped into a multi-valued sub-element, the type of which is defined as a global complex type *ct'* and added into *GT*.

Inheritance: Each *rel* ∈ C_{cdm}.*REL* defined in a class C_{cdm} that inherits another class C'_{cdm} where *rel.relType* := *'inherits'*, is mapped as an inheritance. A complex type *ct* corresponding to $C_{cdm}$is defined as an extension of its complex type *ct'* corresponding to C'_{cdm}. This realizes an XML inheritance between *ct* and *ct'*, where *ct.base* := C'_{cdm}.*cn*.

Example 2: Figure 2 shows a portion of the XML Schema document generated from the CDM given in Table 1, according to the rules presented in Section 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name= "XMLSchema">
<xs:complexType><xs:sequence>
<xs:element name="Dept" type="Dept_t" maxOccurs= "unbounded"/>
<xs:element name= "Hourly_emp" type= "Hourly_emp_t"
      maxOccurs = "unbounded"/>
<xs:element name= "Salaried_emp" type="Salaried_emp_t"
      maxOccurs = "unbounded"/>
<xs:element name="Proj" type="Proj_t" maxOccurs= "unbounded"/>
</xs:sequence></xs:complexType>
<xs:key name= "salaried empEnoPK">
<xs:selector xpath= ".//Salaried_emp"/>
<xs:field xpath= "eno"/>
</xs:key>
...
<xs:keyref name= "projDnumFK" refer= "deptDnoPK">
<xs:selector xpath= ".//Proj"/>
<xs:field xpath= "dnum"/>
</xs:keyref>
...
</xs:element>
 ...
<xs:complexType name= "Emp_t"> abstract= "true"
<xs:sequence>
<xs:element name= "ename" type= "xs:string"/>
<xs:element name= "eno" type= "xs:int"/>
<xs:element name= "bdate" type= "xs:date" minOccurs= "0"/>
<xs:element name= "address" type= "xs:string"/>
<xs:element name= "spreno" type= "xs:int" minOccurs= "0"/>
<xs:element name= "dno" type= "xs:int"/>
<xs:element name= "hasKids" type= "Kids_t"
    minOccurs= "0"   maxOccurs= "unbounded"/>
<xs:element name="Projects" type="Project_t"
maxOccurs="unbounded"/>
</xs:sequence></xs:complexType>
<xs:complexType name = "Salaried_emp_t"><xs:complexContent>
<xs:extension base= "Emp_t"><xs:sequence>
<xs:element name = "salary" type= "xs:int" minOccurs= "0"/>
</xs:sequence></xs:extension>
</xs:complexContent></xs:complexType>
<xs:complexType name = "Kids_t"><xs:sequence>
<xs:element name = "kname" type= "xs:string"/>
<xs:element name = "sex" type= "xs:string" minOccurs= "0"/>
</xs:sequence></xs:complexType>
...
</xs:schema>
```

Figure 2. Sample output XML Schema

5. EXPERIMENTAL STUDY

The proposed solution is implemented as a prototype to realize its algorithm and demonstrate its effectiveness and validity. An experiment conducted to evaluate our approach by examining the differences between the source RDB and the XML documents generated by the prototype. The experiment tests schema information preservation by comparing target schema generated from the prototype with that translated from the same source schemas using existing manual mapping technique, i.e., [3], which gives the user an opportunity to use all features of an XML model and its conceptual schemas, resulting in well-designed schemas. The evaluation includes comparisons of the schema structures, data semantics and integrity constraints.

Elmasri and Navathe [3] described a general algorithm for mapping an EER into an RDB schema and then into XML Schema using a database called **Company**. We used the **Company** RDB as input for our prototype, aiming to generate an XML document from it. The XML Schema file generated from this database is comparable to the XML Schema file mapped from [3]. The schema generated by our prototype is given in Figure 3. The two XML schemas generated by both approaches can be found in [3, 10]. The description of both schemas, including elements and their types, occurrences, keys and attributes, are specified similarly in both approaches. Elements are specified with a type attribute so that the structure of the elements are defined separately. In terms of semantic information preservations, it was found that both schemas were comparable. However, our prototype maps more precisely the attributes and their types and whether each attribute is optional or required.

The resulting schemas show our algorithm and existing manual algorithm to be equivalence-preserving translations. Furthermore, our proposal is a fully-automatic approach and has the ability to generate more accurate target schemas. Therefore, the CDM, which preserves an enhanced structure of an existing RDB, is translatable into the target schema. The algorithm is correct in the sense that it has preserved the original information of the RDBs. Many implicit semantics have been converted from an RDB into the target database, e.g., association, aggregation and inheritance. Moreover, the main type of constraints that can be extracted from an RDB, including key constraints, constraints on NULLs and entity and referential integrity constraints, are all translated explicitly into the equivalent target schema.

```
<xs:complexType name = "Employee_t">
<xs:sequence>
<xs:element name = "fname" type= "xs:string"/>
<xs:element name = "minit" type= "xs:string" minOccurs= "0"/>
```

```
<xs:element name = "lname" type= "xs:string"/>
<xs:element name = "ssn" type= "xs:int"/>
<xs:element name = "bdate" type= "xs:date" minOccurs= "0"/>
<xs:element name = "address" type= "xs:string" minOccurs= "0"/>
<xs:element name = "salary" type= "xs:int" minOccurs= "0"/>
<xs:element name = "superssn" type= "xs:int" minOccurs= "0"/>
<xs:element name = "dno" type= "xs:int"/>
<xs:element name="hasDependent" type= "Dependent_t"
    minOccurs= "0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
...
<xs:complexType name = "Dependent_t">
<xs:sequence>
<xs:element name = "dependent_name" type= "xs:string"/>
<xs:element name = "sex" type= "xs:string"/>
<xs:element name = "bdate" type= "xs:date" minOccurs= "0"/>
<xs:element  name = "relationship" type= "xs:string"
minOccurs= "0"/>
</xs:sequence>
</xs:complexType>
```

Figure. 3 Fragment of XML Company schema generated by our prototype.

6. CONCLUSION

This paper contributes a solution to the problem of translating RDBs into XML schema documents. The approach is beneficial compared to existing work as it generates the XML schema and exploiting the range of powerful features provided by XML Schema standard. A prototype has been developed to realize the algorithm of the solution, which is also validated by comparing results from the input and output schemas. We have conducted an experiment to evaluate our approach by examining the differences between the source RDB and the XML Schema generated by the prototype. The inputs and outputs of the prototype are evaluated in terms of schema structures, data semantics and integrity constraints. The experimental results obtained from both databases have been analyzed and found that both sets of results were identical. Therefore, we conclude that the source and target database schemas are equivalent. Moreover, the results obtained demonstrate that the solution, conceptually and practically, is feasible, efficient and correct.

7. REFERENCES

[1] Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E., and Subramanian, S.2000. XPERANTO: Publishing object-relational data as XML. In *Proceedings ofWebDB*, pp. 105-110.

[2] Dobbie, G., Wu, X., Ling, T., and Lee, M. 2000. ORA-SS. Object-relationship attribute model for semi-structured data. Technical Report TR21/00, National University of Singapore, Department of Computer Science.

[3] Elmasri, R. and Navathe, S. B. 2010. *Fundamentals of database systems (6th Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA.

[4] Lee, D., Mani, M., Chiu, F., and Chu, W. W. 2001. Nesting-based relational to XML schema translation. In *Proceedings ofWebDB*, pp. 61-66.

[5] Kappel, G., Kapsammer, E., and Retschitzegger, W.2004. Integrating XML and relational database systems. In *World Wide Web*, vol. 7(4), pp. 343-384.

[6] Fong, J. and Cheung, S. K.2005. Translating relational schema into XML schema definition with data semantic preservation and XSD graph. In*Information & SoftwareTechnology*, vol. 47(7), pp. 437-462.

[7] Fong J. and Shiu H. 2012. An Interpreter approach for exporting relational data into XML documents with structured export markup language. In*Journal of Database Management*, vol. 23(1), pp. 49-77.

[8] Funderburk, J. E., Kiernan, G., Shanmugasundaram, J., Shekita, E. J., and Wei, C.2002. XTABLES: Bridging relational technology and XML. In*IBM Systems Journal*, vol. 41(4), pp. 616-641.

[9] Maatuk, A., Ali, M. A. and Rossiter, N.2010. Semantic enrichment: The first phase of relational database migration, Tarek Sobh (ed). In*Innovations and Advances in Computer Sciences and Engineering*, Springer, pp. 373-378.

[10] Maatuk, A. 2009.*Migrating Relational Databases into Object-based/XML Databases*, PhD Thesis, Northumbria University, UK.

[11] Valentine, C., Tittel, E., and Dykes, L.2002.*XML Schemas*. SYBEX Inc., Alameda, CA, USA.

[12] Vela, B. and Marcos, E.2003. Extending UML to represent XML schemas. In Eder, J. and Welzer, T., editors, *CAiSE Short Paper Proceedings*, vol. 74 of *CEURWorkshop Proceedings*.

[13] Wang, C., Lo, A., Alhajj, R. and Barker, K.2005. Novel approach for reengineering relational databases into XML. In *ICDE Workshops*, pp. 1284.

[14] W3C. 2014. World Wide Web Consortium (W3C). XML Schema. [Online]. Available: http://www.w3.org/XML/Schema.

[15] Maatuk, A., Ali, M. A. and Rossiter, N. 2011. Re-engineering relational databases: the way forward. In *Proceedings of the Int. Conf. on Intelligent Semantic Web-Services and Applications* (ISWSA '11). ACM, New York, 10 pages. DOI=http://doi.acm.org/10.1145/1980822.1980839