A Framework for Relational Database Migration

Abdelsalam Maatuk, Akhtar Ali, and Nick Rossiter

***Abstract*—The dominance of traditional Relational DataBases (RDBs) and their limitation to support complex structure and user-defined data types provided by object-based/XML databases makes migrating an RDB into object-oriented database, object-relational database and XML an active research area. The problem is how to effectively migrate existing RDBs, as a source, into OODB/ORDB/XML, as targets, and what is the best way to enrich and maintain RDBs' semantics and constraints in order to meet the characteristics of the three targets? Existing work does not appear to provide a complete solution for more than one target database. We tackle this question by proposing a solution for migrating an RDB into the three targets based on available standards. The solution takes an existing RDB as input, enriches its mata data representation with as much semantics as possible, and constructs an enhanced Relational Schema Representation (RSR). Based on the RSR, a Canonical Data Model (CDM) is generated, which captures essential characteristics of target data models suitable for migration. A prototype has been implemented, which migrates a CDM/RDB into object-oriented (ODBG 3.0), object-relational (Oracle 10g) and XML databases.**

I. Introduction

Many organisations have stored their data in RDBs and aspire to take advantage of databases that have emerged more recently, e.g., object-based/XML. Instead of discarding the previous RDB or build non-relational applications on it, it is accepted to convert the old data and applications together into a new environment. However, the question is which of the new databases is it most appropriate to move to? So there is a need for an integrated method that deals with DataBase Migration (DBM) from RDB to Object-Oriented DataBase (OODB)/Object-Relational DataBase (ORDB)/XML in order to provide an opportunity for exploration, experimentation and comparison among the alternative databases. The method should assist in evaluating and choosing the most appropriate target database to adopt for non-relational applications to be developed according to required functionality, performance and suitability, and could help increase their acceptance among enterprises and practitioners. However, the difficulty facing this method is that it is targeting more than one database models and these are conceptually different. There is a lack of a canonical model that can be used as an intermediate stage for schema and data conversion from input (RDB) to various output targets.

Different researches are dedicated in RDB migrations focusing on different areas. Most existing proposals are restricted by a range of assumptions such as a source schema is required to be available for further normalisation to 3rd Normal Form (3NF) before the DBM process can begin [7]. Key-based inclusion dependency is assumed in many proposals with key attribute consistencies and another frequent assumption is that the initial schema is well designed [7], [10], [19]. Earlier models such as Entity Relationship Model (ERM), Extended ERM (EERM) and Object-Modeling Technique (OMT) are assumed in most works as an Intermediate Conceptual Representation (ICR) or target data models [12], [14] whereas others are restricted to a particular product, e.g., Oracle [11]. Several previous approaches fail to maintain all of the data semantics explicitly, e.g., integrity constraints and inheritance, whereas constraints are mapped into class methods [7] or into separate constraint classes [12]. For proof of concept, most proposals have been implemented in one way or another. Only a few attempt data conversion and even those that do have some drawbacks. However, the existing work does not provide a solution for more than one target database or for either schema or data conversion. Besides, none of the previous proposals can be considered as a method for converting an RDB into an ORDB.

In this paper, we propose an integrated method for **MIG**rating an **R**DB into **O**bject-based and **X**ml databases (MIGROX), which is able to preserve the structure and semantics of an existing RDB to generate OODB/ORDB/XML schemas, and to find an effective way to load data into target databases without lose or unnecessary redundancies. The method is superior to the existing proposals as it can produce three different output databases as shown in Fig 1. In addition, the method exploits the range of powerful features that target data models provide such as ODMG 3.0, SQL4, and XML Schema. Due to the heterogeneity among the three target data models, we believe that it is necessary to develop a Canonical Data Model (CDM) to bridge the semantic gap between them and to facilitate the migration process. The CDM should be able to preserve and enhance RDB's integrity constraints and data semantics to fit in with target database characteristics. MIGROX has three phases: Semantic enrichment, schema translation and data conversion. In the 1^{st} phase, the method produces a CDM, which is enriched with an RDB's constraints and data semantics that may not have been explicitly expressed in it. The CDM so obtained is mapped into target schemas in the 2^{nd} phase. The 3^{rd} phase converts an RDB data into its equivalents in the new database environment. System architecture has been designed and a prototype implemented to demonstrate the process, which results successfully in target databases.

This paper is structured as follows. Section II provides an introduction to the semantic enrichment phase. An overview of the schema translation phase is introduced in Section III. Section IV presents the data conversion phase. Section V reviews some results of the MIGROX prototype. A general overview of the related work is presented in Sec-

School of Computing, Engineering & Information Sciences
Northumbria University, Newcastle upon Tyne, UK
{abdelsalam.maatuk,akhtar.ali,nick.rossiter}@unn.ac.uk

tion VI, and Section VII concludes the paper and points to future work.

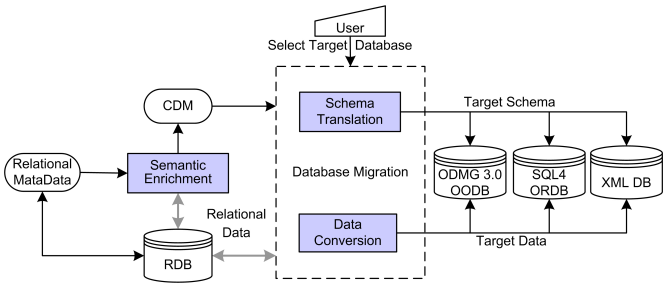

Fig. 1. An Overview of MIGROX

II. Semantic Enrichment

Semantic enrichment is a process of analysing an RDB to understand its structure and meaning, and make hidden semantics explicit. In our approach, the semantic enrichment phase involves the extraction of data semantics of an RDB to be represented in an RSR followed by conversion into a much enriched CDM. This facilitates migration into new target databases without referring to the existing RDB repeatedly. The main benefit from using an RSR and a CDM together is that an RDB is read and enriched once while the results can be used many times to serve different purposes. Fig 2 shows the schematic view of the SE phase.

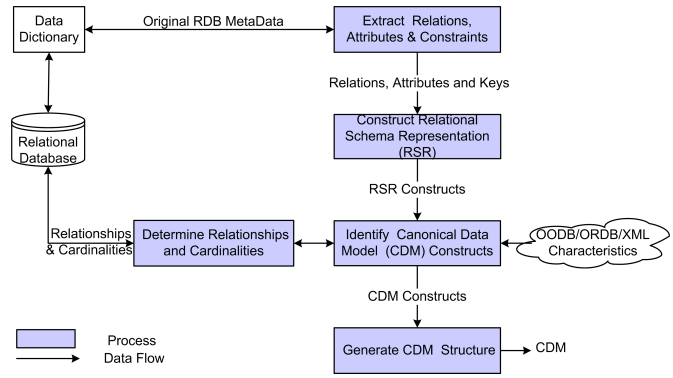

Fig. 2. Schematic View of the Semantic Enrichment Process

The semantic enrichment starts by extracting the basic metadata information about an existing RDB (e.g., relation names and keys) in order to construct an RSR, which is designed in such a way as to ease key matching for RSR's constructs classification. To get the best results, it is preferable that the process to be applied to a schema in 3NF. A relation that is not in 3NF may have redundant data, update anomalies problem or no clear semantics of whether it represents one real world entity or relationship type. The next steps are to identify the CDM constructs based on a classification of RSR constructs, including relationships and cardinalities, which are performed through data access. Lastly, the CDM structure is generated. In databases, essential semantics come with the schema, whereas some semantics might be hidden in application programs. Data semantics can be extracted using a variety of ways such as catalogue (i.e., data dictionary), DataBase Reverse Engineering (DBRE) tools, specified by a user interactively, and obtained from conceptual schemas and design documents. However, conceptual schemas may not be precisely recovered in DBRE from the final logical or physical schemas, and database understanding is easily lost when experienced user(s) are absent or design documents are missing [1]. In modern RDB systems, matadata is usually stored in a catalogue, which can be accessed to get information about a database structure.

A. Extracting Relational Schema Representation (RSR)

In order to produce an integrated source of database semantic information for the purpose of SE, implicit semantics have to be made explicit. Conflicts in naming have to be resolved, and attributes and interrelationships amongst data have to be deduced. In this section, we introduce an RSR, as a representation of an RDB's metadata, to be used as a source of information for CDM generation. Basic information needed to proceed with the SE phase includes relation names and attributes' properties that include attribute names, data types, max length, default values, and whether the attribute is nullable. Moreover, the most important information needed is about the keys including Unique Keys (UKs). We assume that data dependencies are represented by Primary Key (PKs) and Foreign Key (FKs) as for each FK value(s) there are an existing, matched PK value, which can be considered as a value reference. The inverse of an FK is called an Exported Key (EK). EKs play an important role as regards to OO/OR databases, which support bi-directional relationships. The user interaction might be necessary to provide any missing semantics.

Definition 1: In our approach an RDB schema is represented as a set of elements,

RSR := $\{R \mid R := \langle r_n,\ \mathbf{A}_{rsr},\ \mathbf{PK},\ \mathbf{FK},\ \mathbf{EK},\ \mathbf{UK}\rangle\}$, where:

- r_n denotes the name of R.
- $\mathbf{A}_{rsr}$ denotes the set of attributes of R: $\mathbf{A}_{rsr} := \{a \mid a := \langle a_n, t, l, n, d\rangle\}$, where a_n is an attribute name, t is its type, l is data length, n is nullable or not ('y'|'n') and d is a default value if given.
- **PK** denotes R's primary key: $\mathbf{PK} := \{\alpha \mid \alpha := \langle pa, s\rangle\}$, where pa is an attribute name and s is a sequence number in the case of a composite key, however, s is assigned 1 in the case of a single valued key.
- **FK** denotes the set of foreign key(s) of R: $\mathbf{FK} := \{\beta \mid \beta := \langle er,\ \{\langle fa,\ s\rangle\}\rangle\}$, where β represents one FK (whether it is single or composite), er is the name of an exporting (i.e., referenced) relation that contains the referenced PK, fa is a foreign key attribute name, and s is a sequence number.
- **EK** is a set of exported key(s) of R: $\mathbf{EK} := \{\gamma \mid \gamma := \langle ir,\{\langle ea,\ s\rangle\}\rangle\}$, where γ represents one EK, ir is the name of an importing (i.e., referencing) relation that contains the exported attribute name ea (i.e., FK attribute).
- **UK** is a set of unique keys of R: $\mathbf{UK} := \{\delta \mid \delta := \{\langle ua,$

$s\rangle\}\}$, where δ represents one UK, ua is an attribute name and s is a sequence number.

An RSR provides an image of matadata obtained from an existing RDB. However, it provides more semantic information than is readily available in a DD. The main purpose behind constructing an RSR is to read essential metadata into memory outside the database's secondary storage. An efficient RSR construction overcomes the complications that occur during matching of keys in order to classify relations (e.g., strong or weak relation), attributes, e.g., non-key attribute (NK) and relationships, e.g., M:N, inheritance, etc. Each relation R is constructed with its semantics as one element, which is easily identifiable and upon which set theoretic operations can be applied for matching keys. Each of R's elements describes a specific part of R (e.g., $\mathbf{A}_{rsr}$ describes R's attributes). An important advantage of RSR is that it identifies the set **EK**, therefore adding more semantics to an RDB's metadata. The **EK** holds keys that are exported from R to other relations.

Consider the database shown in Fig 3. PKs are *italic* and FKs are marked by "*". Table I gives the RSR constructed for the schema in Fig 3 showing only some relations: emp, salaried_emp, dept and works_on.

EMP

ENAME	*ENO*	BDATE	ADDRESS	SPRENO*	DNO*
Smith	12345	09-Jan-55	31 Hampstead Rd, NE4 8AB	86655	4
Wong	34455	08-Dec-45	16 Hampstead Rd, NE1 7RU	86655	4
Scott	53453	31-Jul-62	4 Northcote St., NE5 5AL	34455	1
Ali	68844	15-Sep-52	49 Hill Street, RG1 2NU	34455	1
Borg	86655	10-Nov-27	162 London Road, OL1 4BX	null	2
Wallace	54321	20-Jun-31	91 St James Gate, NE1 4BB	86655	2
Ally	98798	29-Mar-59	6 Blandford Square, NE1 4HZ	54321	3
Zelaya	98877	19-Jul-58	30 Ripon Gardens, NE2 1HN	54321	3

PROJ

PNAME	*PNUM*	PLOCATION	DNUM*
Way Station 1	1	Newcastle	3
Way Station 2	2	London	3
Way Station 3	3	Reading	3
Salford House	4	Salford	2
4 Times Square	5	Reading	1
Newbenefits	6	Salford	4

DEPT

DNAME	*DNO*	MGR*	STARTD
Accounts	1	86655	19-Jun-71
Administration	2	54321	01-Jan-85
Research	3	34455	22-May-78
Finance	4	12345	06-Oct-05

SALARIED_EMP

*ENO**	SALARY
86655	55000
54321	43000
98798	25000
98877	25000

WORKS_ON

*ENO**	*PNO**
12345	1
34534	1
34455	1
12345	2
34534	2
34455	2
68844	3
34455	3
54321	4
98877	4
98798	4
98798	5
54321	5
86655	5
98877	6
98798	6

KIDS

*ENO**	*KNAME*	SEX
12345	Alice	F
12345	Elizabeth	F
12345	Michael	M
34455	Alice	F
34455	Joy	F
34455	Ally	M
54321	Scott	M

DEPT_LOCATIONS

*DNO**	*LOCATION*
1	Reading
2	Salford
2	Newcastle
3	London
4	Reading

HOURLY_EMP

*ENO**	PAY_SCALE
12345	3
34455	4
53453	2
68844	3

Fig. 3. Sample company database

B. Generating Canonical Data Model (CDM)

This subsection presents a formal definition of CDM. CDM is a good source of valuable semantics, enriched and well organized data model so that it can be converted easily and flexibly to any target models in DBM. Besides taking into account the target model characteristics, it keeps all data semantics that might be extracted from an RDB and integrity constraints imposed on it. Moreover, it represents a key mediator for converting an existing RDB data into target databases based on the structure and the concepts of target model through instance conversion rules. CDM facilitates reallocation of attribute values in an RDB to the appropriate values in a target database. Based on the CDM definition, target attributes that represent relationships among classes are materialized into references or changed into other domain.

In this study the CDM is designed to upgrade the semantic level of RDB and to play the role of an intermediate stage for DBM from RDB to OODB/ORDB/XML acting on both levels: schema translation and data conversion. It represents explicit as well as implicit semantics of an existing RDB. Explicit semantic include relation and attribute names, keys, etc.; implicit semantic include classification of classes and attributes, and relationship names, types, cardinalities and inverse relationships. Its constructs are classified to facilitate the translation to complex target objects in reasonable way instead of flat one to one and complicated clustering conversions. Through the CDM, target databases can be obtained well-structured without proliferation of references and without unnecessary redundancy. However, its richness may not be fully exploited due to the relatively limited expressiveness of the input RDB. For instance, object-based models encapsulate static (i.e., attributes and relationships) and dynamic aspects (i.e., methods) of objects. However, dynamic aspects get less attention in CDM compared to static aspects because an RDB does not support methods attached to tables.

CDM has three concepts: class, attribute and relationship. The model can be seen as an independent model, which embraces object oriented concepts with rich semantics that cater for OR and XML data models. However, the CDM is independent of an RDB from which it has taken semantics as well as any of the target databases to which it could be converted. It is enriched by semantics from an RDB model such as PKs, FKs, attributes max length, uniqueness, etc. In order to express as much semantics as possible, the model has taken into consideration features that are provided by object-based and XML models such as association, aggregation and inheritance. It provides non-OODB key concepts (i.e., FKs, null and UKs) and explicitly specifies that the attributes and cardinalities are optional or required. Relationships are defined in CDM in a way, which facilitates the extracting and transforming of data in the data conversion phase including defining and linking objects using user defined identifiers. Real world entities, multi-valued and composite attributes, and relationships are all represented as classes in CDM.

Definition 2: 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, given a classification cls and whether it is abstract abs or not. Each C has a set of attributes A_{cdm}, a set of relationship Rel and a set of UKs UK.

- *Classification* (cls): A class C is classified into different kinds of classes, which facilitate its translation into target schema, where:
  cls := Regular Strong Class (RST) | Secondary (inherited) Strong Class (SST) | Subclass (SUB) | Secondary (inherited) Subclass (SSC) | Regular (M:N relationship class without attributes) Relationship Class

TABLE I
RESULT OF RSR CONSTRUCTION

r_n	$\mathbf{A}_{rsr}$					**PK**		**FK**			**EK**			**UK**	
	a_n	t	l	n	d	pa	s	er	fa	s	ir	ea	s	ua	s
emp	eno	int	25	n		eno	1	emp	spreno	1	salaried_emp	eno	1		
	ename	char	40	n				dept	dno	1	hourly_emp	eno	1		
	bdate	date		y							works_on	eno	1		
	address	char	40	y							dept	mgr	1		
	spreno	int	25	y							kids	eno	1		
	dno	int		n											
salaried_emp	eno	int	25	n		eno	1	emp	spreno	1	dept	mgr	1		
	salary	int		y											
dept	dno	int		n		dnum	1	emp	mgr	1	emp	dno	1		
	dname	char	40	n							proj	dnum	1	mgr	1
	mgr	int	25	n							dept_locations	dno	1		
	startd	date		y											
works_on	eno	int	25	n		eno	1	emp	eno	1					
	pno	int		n		pno	2	proj	pno	1					

(RRC) | Secondary Relationship Class (SRC), i.e., referenced RRC, M:N relationship with attributes or n-ary relationships, n>2 | Multi-valued Attribute Class (MAC) | Composite Attribute Class (CAC) | Regular Component (in relationship with other classes rather than its whole) Class (RCC).

- *Abstraction* (*abs*): It is important for a superclass to check whether all of its objects are inherited or not. A superclass is abstract (i.e., $abs :=$ `true`) when all its objects are members of its subtype objects. Instances of an abstract type cannot appear in database extension but are subsumed into or by instances of its subtypes. A class is not abstract (i.e., $abs :=$ `false`) when all (or some of) its corresponding RDB table rows are not members of other subtable rows.
- *Attributes* (A_{cdm}): A class C has a set of attributes of primitive data type.
  $A_{cdm} := \{a \mid a := \langle a_n, t, tag, l, n, d\rangle\}$, where each attribute a has a name a_n, data type t and a tag, which classifies attributes into a non-key 'NK', 'PK', 'FK' or both primary and foreign key 'PF' attribute. Each a can have a maximum length l, may have a default d value whereas n indicates that a is nullable or not ('y'|'n').
- *Relationships* (*Rel*): A class C has a set of relationships Rel. Each relationship $rel \in Rel$ between a class C and another 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', $dirAs$ denotes a set containing the attribute names representing the relationship from C' side, $invAs$ denotes a set of inverse attribute names representing the inverse relationship from C side, and c is a relationship cardinality constraint. $RelType$ can have the followings values: '*associated with*' for association, '*aggregates*' for aggregation, and '*inherits*' or '*inherited by*' for inheritance. Cardinality c is defined by $min..max$ notation to indicate the occurrence of C' object(s) within C objects, where min is a minimum cardinality and max is a maximum cardinality. Querying (or examining) data in a complete database is used to extract cardinality constraints. Querying data instances may not return the semantics of an existing RDB if data is incomplete as different database states give different cardinalities. Based on c, C''s object(s) can be single-valued or set-valued.

Using key matching, RSR relations and their attributes are classified, relationships among relations are identified and their cardinalities are determined and translated into equivalents in the CDM. Abstraction of each class in CDM is checked. We assume that, in RDB, the kinds of relations are identified and relationships are represented by means of PKs/FKs. For example, a weak entity/relation is identified when a PK of a relation is a superset of FKs and the to-one relationship is determined when an FK refers to a PK. Other representations may lead to different target constructs. That is a relation R is a strong relation if its PK is not fully or partially composed of any FKs. Similarly, R is a weak relation if its PK is partially composed of a PK of another strong relation. In addition, in EERM, an inheritance relationship is represented using generalization/specialzation, which have different types of specification. However, such types of specialization can not be represented directly in relational data models. There are many alternative ways to model inheritance in relational data models [5]. The most common alternative represents inheritance as one relation for a superclass and one relation for every subclass. The superclass is represented by a relation R with its key and attributes, where $R(pk, a_1,..., a_n)$, $A(R) := \{pk, a_1..., a_n\}$ and $PK(R) := pk$. Each subclass S with its attributes is represented by relation $S(pk,$ attributes of $S)$ and $PK(S) := pk$. MIGROX assumes this alternative because it is based on PKs/FKs matching, and without user involvement it would not be possible to automatically identify other alternatives of inheritance. The main idea in inheritance is that a supertype is inherited by one or more other subtypes. A subtype can be inherited by other subtypes. Moreover, a subtype may inherit from more than one other supertypes, i.e., multiple-inheritance. However, as ODMG 3.0, SQL4 and XML Schema do not allow a concrete subtype to have more than one concrete supertype, multiple-inheritance is not supported in MIGROX. Therefore, each superclass in CDM can be inher-

ited by one or more subclasses, but a subclass can have only one superclass.

Consider the RSR shown in Table I, Fig 4 shows the resulting CDM, generated from the RSR and RDB in an easy to follow format hiding the deeply nested structure of CDM classes. The CDM shows only `EMP` and `DEPT` classes. For instance, the CDM's class, `EMP`, which is SST, has attributes: `ename`, `eno`, `bdate`, `address`, `spreno` and `dno`. Other properties (e.g., attributes' types, tags, default values) are not shown for the sake of space. The class `EMP` is '*associated with*' classes: `DEPT`, `WORKS_ON` and with itself. Moreover, it '*aggregates*' `KIDS` class and '*inherited by*' `SALARIED_EMP` and `HOURLY_EMP` classes. Cardinalities are determined for each class. Relationships defined in each class as: *RelType* $\{invAs \longleftrightarrow dirC(dirAs)_c\}$ ($\longleftrightarrow$ indicates bidirectional association and $\rightarrow$ indicates unidirectional aggradation).

```
EMP^SST_abs [A_cdm := {ename, eno, bdate, address, spreno, dno},
Rel := {associated with{dno ⟷ DEPT(dnum)1..1,
eno ⟷ DEPT(mgr)0..1, spreno ⟷ EMP(eno)1..1,
eno ⟷ EMP(spreno)0..*, eno ⟷ WORKS_ON(eno)1..* },
aggregates{ eno → KIDS(eno)0..*},
inherited by {SALARIED_EMP, HOURLY_EMP}}]

DEPT^RST [A_cdm := {dname, dnum, mgr, startd},
Rel := {associated with {mgr ⟷ EMP(eno)1..1,
dnum ⟷ EMP(dno)1..*, dnum ⟷ PROJ(dnum)1..*},
aggregates {dno → DEPT_LOCATIONS(dno)1..*}}]
```

Fig. 4. Sample generated CDM schema

III. Schema Translation

The Schema Translation phase aims at translating CDM, produced from the semantics enrichment phase, into its equivalent targets schemas as shown in Fig 5. Target schemas hold equivalent semantics to that of an existing RDB, which are enhanced and preserved in CDM. Three sets of translation rules are designed for mapping CDM into target schemas. Algorithms are developed for producing each target schema according to these rules. In this section, we define target schemas, which satisfy ODMG 3.0, SQL4 and XML Schema standards and introduce the schema translation phase.

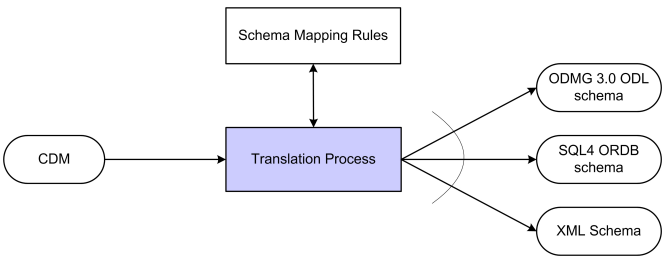

Fig. 5. Schematic View of translating CDM into target schemas

A. Target Models

We first briefly define the output target models for the ST phase. Translating these models defined here to the actual schema definition languages is straightforward.

Definition 3: A target ODMG 3.0 schema is defined as a set of classes,

$OOschema := \{C_{oo} \mid C_{oo} := \langle c_n, spr, k, A_{oo}, Rel_{oo}\rangle\}$, where c_n is a name of a class C_{oo}, spr is the name of its superclass, k is its primary key, A_{oo} is a set of its attributes of simple or complex data type and Rel_{oo} is a set of relationship types in which C_{oo} participates. The sets A_{oo} and Rel_{oo} are defined as follows:

- $A_{oo} := \{a_{oo} \mid a_{oo} := \langle a_n, t, m\rangle\}$, where a_n is a name of an attribute a_{oo}, t is its data type, which can be primitive (e.g., string), user-defined constructed (e.g., **struct**) or object-based (e.g., class), and m denotes whether a_{oo} is single-valued ('sv') or collection-valued ('cv'); and
- $Rel_{oo} := \{rel_c \mid rel_c := \langle rel_n, dirC_n, m, invRel_n\rangle\}$, where rel_n is the name of the relationship rel_c, $dirC_n$ is the name of the referenced class, and $invRel_n$ is the name of the inverse relationship.

Definition 4: A target SQL4 ORDB schema is represented as $ORschema := \{UT, TT, UK_{or}\}$, where UT is a set of User Defined Types (UDTs), TT is a set of typed tables and UK_{or} is a set of UKs. The sets UT and TT are defined as follows:

- $UT := \{udt \mid udt := \langle ut_n, s_{ut}, A_{ut}, uoid\rangle\}$, where ut_n is the name of the type udt, s_{ut} is the supertype name of udt, A_{ut} is a set of attributes and $uoid$ is a user defined identifer of udt;
  - $A_{ut} := \{a_{ut} \mid a_{ut} := \langle a_n, t, m, n, d\rangle\}$, where a_n is a name of an attribute a_{ut}, t is its data type, which can have primitive, user-defined constructed or ref-based, m denotes whether a_{ut} is a single-valued ('sv') or a collection-valued ('cv'), d is a default value in case of primitive and n denotes whether a_{ut} accepts nulls or not; and
- $TT := \{tt \mid tt := \langle tt_n, ut_n, s_{tt}, pk\rangle\}$, where tt_n is the name of a typed table tt, ut_n is a udt's name that tt is defined based upon, s_{tt} is its supertable's name and pk is the PK of tt.

Definition 5: A target XML Schema is represented as $XMLschema := \{Root, GT\}$, where:

Root is a global element declared under the schema with its direct local subelements and constraints representing the XML document tree, and GT is a set consisting of global complex types, which are defined to be referenced as types of subelements that are declared within the *Root* or by other defined global complex types. The *Root* and the set GT are defined as follows:

- $Root := \{root_n, RT, PK_x, FK_x, UK_x\}$, where *Root* has a name $root_n$, a type RT and three sets of identity-constraints PK_x, FK_x and UK_x;
  - RT represents the *Root*'s local complex type that involves a set of local sub-element le declarations: $RT := \{le \mid le := \langle e_n, e_t, nim, max\rangle\}$, where e_n is the name of le, e_t is its type, and min and max are its minimum and maximum occurrences. The type of each subelement e_t is defined globally under the schema in the set GT;

- PK_x is a set of primary keys for subelements defined in the *Root*: $PK_x := \{pk \mid pk := \langle pk_n, selector, PKfield\rangle\}$. Each primary key has a constraint name pk_n, element set *selector* as scope for the key to be defined in, and a set of related sub-elements that are selected to be unique $PKfield$;
- FK_x is a set of foreign keys, where $FK := \{fk \mid fk := \langle fk_n, ref, selector, FKfield\rangle\}$. Each foreign key has a constraint name fk_n, an element set *selector*, a reference constrains name $refer$ that refer to a matched PK constrain name, and a set of related subelements $FKfield$;
- UK_x is a set of unique keys, where $UK := \{uk \mid uk := \langle uk_n, selector, UKfield\rangle\}$. Each unique key has a constraint name uk_n, element set scope *selector*, and a set of related subelements selected to be unique $UKfield$; and

• $GT := \{CT \mid CT := \langle ct_n, base, LE\rangle\}$, where ct_n is the name of a complex type CT, *base* is its a supertype's name (if it is derived from other type), and LE is a set of elements that declared locally within CT. $LE := \{le \mid le := \langle e_n, e_t, nim, max\rangle\}$ as defined as for *Root*'s type; however, data type of local elements $le \in LE$ can be built-in data type (e.g., string) or predefined complex type (e.g., dependent complex type).

B. Translation Process

Given a CDM, the schema translation phase starts by asking the user to determine which target is to be produced. Then, an appropriate set of rules is implemented to map the CDM into equivalent constructs in the target schema. Each rule maps a specific construct, e.g., class or attribute. By using CDM constructs classification, we can identify their equivalents in target schema definition language. Based on *cls*, each main CDM class C is translated into target type, where $C.cls \neq$ ("MAC" | "CAC" | "RRC"). The type is defined under its superclass if $C.cls$:= "SUB" or "SSC". However, MAC and CAC classes are mapped into multi-valued and composite (e.g., **struct**) attributes respectively, and RRC classes are mapped into an M:N relationship in which a pair of 1:M relationship is defined in each of the target types that participate in the relationship. Attributes $C.A_{cdm}$ are translated into equivalents with the same names as that of CDM and their types are converted according to target data types. Keys are specified when attributes are tagged with 'PK'. The type of target relationship and its multiplicity are determined by the classification of a CDM class C' related to the class C being translated and the properties of each relationship *rel* defined in C, where $rel \in C.Rel$, e.g., $rel.RelType$, $rel.c$. Each *rel* is translated into an equivalent target association, aggregation or inheritance. Target relationship names are generated by concatenating $dirC$ with $dirAs$, and $C.cn$ with $invAs$, e.g., `dept_mgr` and `emp_eno` in Fig 6. Relationship cardinality $rel.c$ is mapped into single-valued when $rel.c$:= (0..1 | 1..1) or collection-valued otherwise. The OODB and ORDB schemas corresponding to the CDM in Fig 4 are shown in Fig 6 (ODMG 3.0 ODL) and Fig 7 (Oracle 10_g), respectively. The XML Schema is provided in Fig 8.

```
class emp (extent emps, key eno) {
attribute string ename; attribute number eno;
attribute date bdate; attribute string address;
attribute set<struct kids{string kname, char sex;}> kids_eno;
relationship dept dept_mgr inverse dept::emp_eno;
relationship set<emp> emp_spreno inverse emp::emp_eno;
relationship dept dept_dno inverse dept::emp_dno;
relationship emp emp_eno inverse emp::emp_spreno
relationship set<proj> proj_pnum inverse proj::emp_eno;};
class hourly_emp extends emp (extent hourly_emps){attribute
number pay_scale;};
class salaried_emp extends emp (extent salaried_emps){attribute
number salary;};
class dept (extent depts, key dno) {
attribute string dname; attribute number dno;
attribute date startd; attribute set<string> dept_locations_dno;
relationship set<emp> emp_dno inverse emp::dept_dno;
relationship set<proj> proj_dnum inverse proj::dept_dno;
relationship emp emp_eno inverse emp::dept_mgr;};
class proj (extent projs, key pnum) {
attribute string pname; attribute number pnum; attribute string
plocation;
relationship set<emp> emp_eno inverse emp::proj_pnum;
relationship dept dept_dno inverse dept::proj_dnum;};
```

Fig. 6. Sample Output OODB schema

IV. Data Conversion

The Data Conversion phase concerns converting existing RDB data to the format defined by the target schema. Data stored as tuples in an RDB are converted into complex objects/literals in object-based databases or elements in XML document. We propose using CDM to guide the conversion process. Data coversion is performed in three steps as shown in Fig 9. Firstly, the RDB relations' tuples are extracted. Secondly, these data are transformed (converted) to match the target format. Finally, the transformed data are loaded into text files suitable for bulk loading in order to populate the schema generated earlier during the ST phase. Since relationships in object-based databases are reference-based, the process is accomplished in two separate passes. In the first pass, each RDB relations' tuples comprising of non-FK attributes are converted into equivalent target format in order to define objects. In the second pass, the initial object defined in the first pass are linked using FK values extracted from each RDB relation's tuples based on relationships defined in the target schema. Objects' user-defined identifers *uoid*s[1] are extracted by concatenating the class name with the PK data extracted from corresponding RDB table. Similarly, object-based relationships are established using *uoid*s extracted from CDM relationship attributes, i.e., $dirAs$ and $invAs$ data. However, relationships among XML elements are established by *key*/*keyref* constraints specified in XML schema document. Each target database's data are generated using a set of data instance conversion rules. We have developed an algorithm for integrating the rules for each target database. The algorithm generates the target data in text files as initial objects' files and relationships files. Sets of customised SQL queries are embedded in these al-

[1] i.e., surrogate OID, which will be translated by the system into a physical OID during object loading

```
create type emp_t
create type proj_t
create type dept_locations_t as object (location varchar2(20));
create type dept_locations_ntt as table of dept_locations_t;
create type kids_t as object (kname varchar2(20),sex char(1));
create type kids_ntt as table of kids_t;
create or replace type emp_ntt as table of ref emp_t;
create or replace type proj_ntt as table of ref proj_t;
/
create or replace type dept_t as object (
dname varchar2(20), dno number, startd date, dept_locations_dno
dept_locations_ntt, emp_dno emp_ntt, proj_dnum proj_ntt, emp_eno
ref emp_t) not final;
create or replace type emp_t as object (
ename varchar2(20), eno number, bdate date, address
varchar2(30), dept_mgr ref dept_t, emp_spreno emp_ntt, kids_eno
kids_ntt, proj_pnum proj_ntt, dept_dno ref dept_t, emp_eno ref
emp_t) not final;
create type proj_t as object (
pname varchar2(20), pnum number, plocation varchar2(20), emp_eno
emp_ntt, dept_dno ref dept_t) not final;
create type hourly_emp_t under emp_t (pay_scale number) final;
create type salaried_emp_t under emp_t (salary number) final;

create table dept of dept_t
nested table dept_locations_dno store as dept_locations_dnoNT
nested table emp_dno store as emp_dnoNT nested table proj_dnum
store as proj_dnumNT;
create table hourly_emp of hourly_emp_t
nested table emp_spreno store as emp_spreno_hourly_empNT nested
table kids_eno store as kids_eno_hourly_empNT nested table
proj_pnum store as proj_pnum_hourly_empNT;
create table proj of proj_t
nested table emp_eno store as emp_enoNT;
create table salaried_emp of salaried_emp_t
nested table emp_spreno store as emp_spreno_salaried_empNT nested
table kids_eno store as kids_eno_salaried_empNT nested table
proj_pnum store as proj_pnum_salaried_empNT;
```

Fig. 7. Sample Output ORDB schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:annotation>
<xs:documentation xml:lang ="en"> Generated XML Schema
</xs:documentation>
</xs:annotation>
<xs:element name= "XMLSchema">
<xs:complexType>
<xs:sequence>
<xs:element name= "dept" type= "dept_t" maxOccurs =
"unbounded"/>
<xs:element name= "emp" type= "emp_t" maxOccurs = "unbounded"/>
<xs:element name= "hourly_emp" type= "hourly_emp_t" maxOccurs =
"unbounded"/>
<xs:element name= "proj" type= "proj_t" maxOccurs =
"unbounded"/>
<xs:element name= "salaried_emp" type= "salaried_emp_t" maxOccurs
= "unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:key name= "empenoPK">
<xs:selector xpath= ".//emp"/>
<xs:field xpath= "eno"/>
</xs:key>
...
<xs:keyref name= "empdnoFK" refer= "deptdnoPK">
<xs:selector xpath= ".//emp"/>
<xs:field xpath= "dno"/>
</xs:keyref>
...
</xs:element>
<xs:complexType name= "dept_t">
<xs:sequence>
<xs:element name= "dname" type= "xs:string"/>
<xs:element name= "dno" type= "xs:int"/>
<xs:element name= "mgr" type= "xs:int" minOccurs= "0"/>
<xs:element name= "startd" type= "xs:date" minOccurs= "0"/>
<xs:element name= "dept_locations_dno" type= "xs:string"
maxOccurs= "unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name= "emp_t">
<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= "kids_eno" type= "kids_t" minOccurs= "0"
maxOccurs= "unbounded"/>
<xs:element name= "proj_pnum" type= "proj_pnum_t" maxOccurs=
"unbounded"/>
</xs:sequence>
</xs:complexType>
...
</xs:schema>
```

Fig. 8. Sample Output XML Schema

gorithms to extract the desire data from an RDB. Once a query is executed, the result is transformed from flat RDB form to target database format. At last, a conversion program is generated to emact the schema file obtained from the ST phase and the files generated during the DC phase.

Consider the CDM shown in Fig 4 and RDB data given in Fig 3 to input to the algorithm for generating OODB data. One tuple from the salaried_emp RDB table of an employee called "Wallace", which is identified by the PK value 54321 is converted, with its related tuples in other tables, into target equivalents. The output OODB object definition (in LDB[2] syntax) that represents the RDB "Wallace" tuple is shown in Fig 10(a), whereas its relationships is defined in Fig 10(b).

V. Experimental Study

To demonstrate the effectiveness and validity of MIGROX, a prototype has been developed to realize the algorithms outlines in proceeding sections. The algorithms were implemented using Java 1.5 software development kit and Oracle 10_g. The experiment was run on a PC with CPU Pentium IV 3.2 GHz and RAM 1024 MB operated under Windows XP Professional. We used the JDBC API to establish a connection with RDBMS that hold the source RDB. To evaluate scalability and performance of MIGROX, a set of queries have been designed to observe any differences between the source RDB and target databases. Due to limited space, this section presents only two sets of queries for the RDB shown in Fig 3 and one equivalent target database generated by MIGROX (i.e., ORDB). For each query we give a description, an RDB Version (R-Q), and an ORDB Version (OR-Q) and the result of the query. The queries are run on Oracle 10_g to check whether the results are the same or not.

1. Find the name of a department with dno = 4.
   R-Q/OR-Q: *select dname from dept where dno = 4;*
   Result: `Finance`
2. Find names of salaried employees in department 2 who make more than 40000 per year.
   R-Q: *select e.ename from emp e, salaried_emp s where e.dno = 2 and e.eno = s.eno and s.salary >= 40000;*
   OR-Q: *select s.ename from salaried_emp s where s.dept_dno.dno = 2 and s.salary >= 40000;*
   Result:
   `Borg`

[2] http://lambda.uta.edu/lambda-DB/manual/

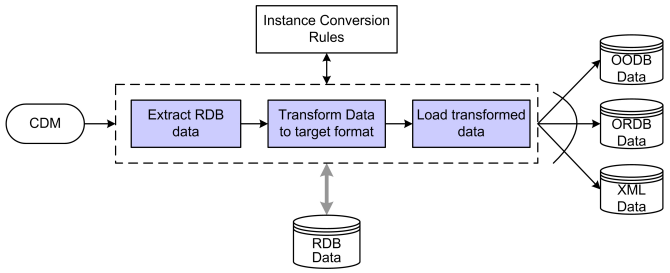

Fig. 9. Schematic view of converting relational data into targets

```
%salaried_emp54321 := persistent hourly_emp (ename: "Wallace",
eno: 54321, bdate: '1931-06-20', address: "91 St James Gate
NE1 4BB", kids_eno: set(struct(kname: "Scott", sex: "M")),
salary: 43000);
```

(a) definition of salaried_emp54321 object

```
salaried_emp54321 -> update()-> dept_dno.add (dept2);
salaried_emp54321 -> update()-> emp_eno.add (salaried_emp86655);
salaried_emp54321 -> update()-> proj_pnum.add (proj4, proj5);
```

(b) relationships among salaried_emp54321 and other objects

Fig. 10. Output OODB data

`Wallace`

3. Find all employees working in 'Accounts'.
   R-Q: *select e.eno, e.ename from emp e, dept d where e.dno = d.dno and d.dname = 'Accounts';*
   OR-Q: *select st.column_value.eno, st.column_value.ename from dept d, table(d.emp_dno) st where d.dname = 'Accounts';*
   Result:
   `34534 Scott`
   `68844 Ali`
4. Find all employees who have kids named "Alice" and "Michael".
   R-Q: *select e.ename from emp e, kids d1, kids d2 where e.eno = d1.eno and e.eno = d2.eno and d1.kname = 'Alice' and d2.kname = 'Michael';*
   OR-Q: *select h.ename from hourly_emp h, table(h.kids_eno) d1, table(h.kids_eno) d2 where d1.kname = 'Alice' and d2.kname = 'Michael';*
   Result: `Smith`
5. Display a list of project names that involve an employee whose name is "Smith".
   R-Q: *select pname from proj p, works_on w, emp e where e.eno = w.eno and w.pno = p.pnum and e.ename = 'Smith';*
   OR-Q: *select pname from proj p, table(p.emp_eno) e where e.column_value.ename= 'Smith';*
   Result:
   `Way Station 1`
   `Way Station 2`

After evaluating the results between the source and the target database, MIGROX is shown to be feasible and efficient as the queries designed for retrieval operations return identical results. Target databases are generated without loss or redundancy of data. Moreover, many semantics can be converted for RDB into the targets, e.g., association, aggregation and inheritance with integrity constraints enforced to the target databases. Some update operations (i.e., insert, delete and modify) are applied on the databases to show that integrity constraints in the RDB are preserved in the target databases. However, we cannot cover automatically referential integrity constraints on REFs that are in nested tables in ORDB because Oracle[3] does not have a mechanism to do so. This integrity could be preserved manually once the schema is generated, e.g., using triggers.

VI. Related Work

In recent years, with the growing importance and benefits provided by object-based and XML databases, there has been a lot of effort on migrating RDBs into the relatively newer technologies [1], [15], [8], [10]. Migration of source RDB into object-based and XML databases is accomplished in the literature for only one target database (e.g., OODD, ORDB or XML). Existing work can be classified into two categories. The first category, which is called Source-to-Target (ST), translates each construct in a source into an equivalent construct in a target database without using an Intermediate Conceptual Representation (ICR) for semantic enrichment. This technique usually results in ill-designed databases as some of the data semantics are ignored. The second category, which is called Source-to-Conceptual-to-Target (SCT), results in well-designed databases due to the amount of data semantics preserved in a conceptual intermediate stage, i.e, ICR.

Inferring conceptual schema from a logical RDB schema has been extensively studied [1], [13], [9]. Such conversions are usually specified by rules, which describe how to derive RDB's constructs (e.g., relations, keys), classify them, and identify relationships among them. Semantic information is extracted by an in-depth analysis of schema, data and queries. Fonkam and Gray present an algorithm for converting RDB schemas into conceptual models [9]. Alhajj proposes semantics extraction by analysing data instances [1]. Petit *et al.* present an approach to extract EERM constructs from an RDB by analysing SQL queries in application programs [13].

Existing work for migrating RDBs into OODBs focus on schema translation using ST techniques [14], [15], [7]. Premerlani and Blaha propose a procedure for mapping an RDB schema into an OMT schema [14]. Fahrner and Vossen propose a method, in which an RDB schema is normalised to 3NF, enriched by semantics using data dependencies and translated into an ODMG-93 schema [7]. Singh *et al.* propose an algorithm for mapping an RDB schema into a corresponding OODB schema based on common attributes factoring [15]. However, constraints are not considered in their approach. Behm *et al.* propose a model, called Semi Object Type (SOT), to facilitate restructuring of schemas during translating an RDB into an OODB [2]. An RDB schema is mapped into SOT schema, which is

[3] http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96594/title.htm

then redesigned and converted into an OODB schema. How to map UML models to ORDBs has been studied not long ago (e.g., [16], [11]), however, the focus has been on the design of ORDBs rather than on migration. Most of existing research on migrating RDBs to XML are following the SCT technique, focusing on generating a DTD schema and data [3], [4], [17]. Some existing work (e.g., [6], [4]) use data dictionaries and assume well-designed RDB (e.g., in 3NF) whereas some others consider legacy RDB (e.g., [18]) for migration into XML documents. Du *et al.* propose a method that employs a model called ORA-SS to support the translation of RDB schema into XML Schema [4].

Although known conceptual models, e.g., ERM and UML may be used as a CDM during DBM, we argue that they do not satisfy the characteristics and constructs of more than one target data model, and do not support data representation. Some important semantics have not been considered in these conceptual models. For instance, ERM does not support inheritance whereas UML should be extended by adding new stereotypes or other constructs to specify ORDB and XML models peculiarities [11], [17]. Several ICR models have been developed for specific applications. However, these models are incapable of capturing diverse characteristics of the three target data models. The SOT model [2] has been designed only for migrating RDBs into OODBs whereas the ORA-SS model [4] has been designed to support semi-structured data models.

VII. Conclusion

This paper contributes a solution to RDB migration, which is superior to the existing proposals as it can produce three different output databases. Besides, it exploits the ranges of powerful features that target data models provide such as ODMG 3.0, SQL4, and XML Schema. A system architecture is designed and a prototype has been implemented, which generated successfully the target databases. The approach has been evaluated by comparing query results. We have designed several experiments that involve running queries on a source RDB and one target database, which is generated by our prototype. We have analysed the results of queries obtained from both databases and found that both set of results were identical. Therefore, we concluded that the source and target databases are equivalents. Moreover, the results obtained demonstrate that the MIGROX solution, conceptually and practically is feasible, efficient and correct. Our future research focus is on data specific manipulation (e.g., update/query) translations and further prototyping to simplify relationship names that are automatically generated.

References

[1] Alhajj, R.: Extracting the Extended Entity-Relationship Model from a Legacy Relational Database. Inf. Syst, vol. 28, pp. 597–618, 2003.

[2] Behm, A., Geppert, A. and Dittrich, K. R.: Algebraic Database Migration to Object Technology. ER'00, pp. 440–453, 2000.

[3] Conrad, R., Scheffner, D., Freitag J. C.: XML Conceptual Modeling Using UML. In 19th Int. Conf. Conceptual Modeling, vol. 1920, pp. 558–571, 2000.

[4] Du, W., Li, M., Tok, L., and Ling, W.: XML Structures for Relational Data. WISE, vol. 1, pp. 151–160, 2001.

[5] Elmasri, R. and Navathe, S. B.: Fundamentals of Database Systems (5th Edition). Addison-Wesley, Inc., 2006.

[6] Lee, D., Mani, M., Chiu, F. and Chu, W. W.: NeT and CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints. CIKM, pp. 282–291, 2002.

[7] Fahrner, C. and Vossen, G.: Transforming Relational Database Schemas into Object-Oriented Schemas According to ODMG-93. In 4th Int. Conf. on Deductive and Object-Oriented Databases, pp. 429–446, 1995.

[8] Fong, J., Wong, H. K. and Cheng, Z.: Converting Relational Database into XML Documents with DOM. Info. & Soft. Tech., vol. 45, pp. 335–355, 2003.

[9] Fonkam, M. M. and Gray, W. A.: An Approach to Eliciting the Semantics of Relational Databases. In 4th Int. Conf. on Advanced Info. Syst. Eng., vol. 593, pp. 463–480, 1992.

[10] Kleiner, C. and Lipeck, U. W.: Automatic Generation of XML DTDs from Conceptual Database Schemas. GI Jahrestagung, vol 1, pp. 396–405, 2001.

[11] Marcos, E., Vela, B. and Cavero, J. M.: A Methodological Approach for Object-Relational Database Design using UML. Soft. and Syst. Modeling, vol. 2, pp. 59–75, 2003.

[12] Narasimhan, B., Navathe, S. B. and Jayaraman, S.: On Mapping ER Models into OO Schemas. In 12th int. Conf. Entity-Relationship Approach, vol. 823, pp. 402–413, 1993.

[13] Petit, J., Kouloumdjian, J., Boulicaut, J. and Toumani, F.: Using Queries to Improve Database Reverse Engineering. In 13th Int. Conf. Entity-Relationship Approach, vol. 881, pp. 369–386, 1994.

[14] Premerlani, W. J. and Blaha, M. R.: An Approach for Reverse Engineering of Relational Databases. Commun. ACM, vol. 37, pp. 42–49, 1994.

[15] Singh, A., Kahlon, K. S., Singh, J., Singh, R., Sharma, S. and Kaur, D.: Mapping Relational Database Schema to Object-Oriented Database Schema. Int. Conf. on Computational Intelligence, pp. 153–155, 2004.

[16] Urban, S. D., Dietrich, S. W. and Tapia, P.: Succeeding with Object Databases: Mapping UML Diagrams to Object-Relational Schemas in Oracle 8. John Wiley, pp. 29–51, 2001.

[17] Vela, B. and Marcos, E.: Extending UML to Represent XML Schemas. CAiSE Short Paper Proceedings, 2003.

[18] Wang, C., Lo, A., Alhajj, R. and Barker, K.: Converting Legacy Relational Database into XML Database through Reverse Engineering. ICEIS, vol. 1, pp. 216–221, 2004.

[19] Zhang, X., Zhang, Y., Fong, J. and Jia, X.: Transforming RDB Schema to Well-structured OODB Schema. Info. & Soft. Tech., vol. 41, pp. 275–281, 1999.