Concurrent data materialization for Object-Relational database with semantic metadata

Joseph Fong, Ringo Pang, Anthony Fong*, Francis Pang, Kenny Poon
Computer Science Department *Electronic Engineering Department
City University of Hong Kong, Tat Chee Avenue, Hong Kong, email: csjfong@cityu.edu.hk

Abstract
For a company with many databases in different data models, it is necessary to consolidate them into one data model interchangeable and present data in one data model concurrently to users. The benefit is to let user stick to his/her own data model to access database in another data model. This paper presents a semantic metadata to preserve database constraints for data materialization to support user’s view of database on ad hoc base. The semantic metadata can store the captured semantics of a relational or an object-oriented database into classes and stored procedures triggered by events. The stored constraints and data can be materialized into a target database upon user request. The user is allowed to perform the data materialization many times alternatively. The process can provide a relational as well as an object oriented view to the users simultaneously. This concurrent data materialization function can be applied into data warehouse to consolidate heterogeneous database into a fact table in a data model of user’s choice. Furthermore, a user can obtain either a relational view or an object-oriented view of the same dataset of an object-relational database interchangeably.

Keywords: Data materialization, semantic metadata, schema translation, relational database, object-oriented database

1 Introduction
For a company with an object-relational database, it is not clear how a user can obtain a relational view while another user obtains an object oriented view at the same time. Hence, a metadata containing the materialization of different data model must be maintained. Data types in different data model need different access mechanism.

To tackle the problems, a semantic metadata aims to capture the semantics of the source database. The semantic metadata contains different data constraints that the target database needs to implement after the data materialization. The procedure for materializing data into any data model is as follows:

Step 1: Capture data semantics of source database into semantic metadata.
1. Extract semantics from the Source database by examining their data occurrences.
2. Store data semantics into the semantic metadata.
3. Capture source database into sequential files.
4. Store source database into the semantic metadata.

Step 2: Transfer the stored data semantics and data into a target database
1. Stored data semantics are transferred into a target database schema.
2. Stored source database are transformed into a target database.

We need to perform schema translation before data materialization[1]. Data materialization is to transform data into different data models upon user request. As a result, a user can reuse a relational database as an object oriented database or vice versa with the same dataset in the semantic metadata. The process can provide a relational view and an object oriented view of the same database to the users concurrently.

2 Related Works
Schema translation translates schema from one data model into another. [2] proposed to translate ER diagram directly into their self-defined OO model which allowed specification of OO schema in three planes. [3] gave a survey of transformation methodologies and approaches for the various directions which were relevant to conceptual and logical database design. [4] proposed to translate a Relational schema into Class definition by use of five phases: Initialisation, Decomposition, Classification, Generalisation and Identification. [5] proposed a two-phase methodology to translate relational schema to OO schema and performed data migration using schema transformation. [6] proposed heuristic algorithms to approximate types semi-automatically. [7] investigated a RDB schema translation process to recapture lost semantics via an EER model using data search technique. [8] proposed three parallel discovery models based on horizontal, vertical, and matrix database table

[12] described the VAT system for data conversion. [13] presented a translation system, based on schema-matching, aimed at simplifying the intricate task of data conversion. [14] presented a technique for querying and transforming scientific data. [15] described a methodology for schema translation and data conversion from Relational database to Object oriented database. [16][17] implemented a data conversion system with mathematical verification. [18] presented an approach to load large volumes of data, and highlighted factors of reengineering used for repeatable processes in large industrial settings. [19] described a framework for data integration based on OO types hierarchies and late binding. [20] depicted an approach to database interoperation that exploits the semantic information provided by integrity defined on the component databases. [21] rendered a framework for data integration, based on a special class of mediators called Squirrel integration mediators. These mediator supported views that integrate data from multiple data sources, and supported the materialized and virtual approaches, and hybrids of them.

[22] stated that the study of semantics and the reality of computing have a strong relationship. [23] described a methodology to unravel the semantics of conceptual schema. [24] described an algorithm that reduced the database activity considerably. [25] introduced the ER model to capture the semantic and the structure of the data in the conceptual schema. [26] introduced Entity-Category-Relationship Model with various variant called the Extended/Enhanced ER Model (EER). [27] introduced Object Modeling Technique (OMT) to cover the operations and rules applied to the data in addition to semantic data modelling techniques. [28] presented a set of abstraction to build an integrated enterprise model, and suggested framework provided an integration of semantic, pragmatic and non-traditional communication dependencies. [29] gave a definition of the semantics of active database using a structured approach. [30] used the Semantic metadata to capture the semantic of a universal database. [31] described a methodology with verification for schema integration for Object Relational Database System. [32] defined a set of primitive transformations for an EER model and by defining some of the common schema transformations as sequences of these primitive transformations. [33] showed how schema transformation can migrate or warp data, queries and updates between semantically equivalent schemas. Other important contributions in the field include the ones by the group of Paolo Atzeni [34, 35, 36] and the ones by the group of Phil Bernstein [37, 38, 39, 40].

This paper suggests a unique metadata solution for data materialization by allowing concurrent relational and object-oriented views of the same data set upon users’ demands.

### 3 Semantic metadata

Semantic metadata is a metadata model. It is a higher-order synthesis with semantic frame concepts, semantic data modeling concepts, object-oriented concepts and production rules concepts. All abstract data semantics in the conceptual schema can be recognized and mapped into the logical representations. It can represent the object behaviour and the relational functionality. A class in the semantic metadata contains four main parts: headers, attributes, methods and constraints. A general description of semantic metadata is given in table 1.

<table>
<thead>
<tr>
<th>HEADER CLASS</th>
<th>ATTRIBUTE CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Name</td>
<td>Class Name</td>
</tr>
<tr>
<td>/* a unique name in all system */</td>
<td>/* a unique name in all system */</td>
</tr>
<tr>
<td>Key</td>
<td>Attributes Name</td>
</tr>
<tr>
<td>/* an attribute name or by default a class */</td>
<td>/* a unique name in this class */</td>
</tr>
<tr>
<td>Parents</td>
<td>Attribute Type</td>
</tr>
<tr>
<td>/* a list of class names */</td>
<td>/* the data type for the attribute */</td>
</tr>
<tr>
<td>Methods</td>
<td>Default Value</td>
</tr>
<tr>
<td>/* a list of methods */</td>
<td>/* predefined value for the attribute */</td>
</tr>
<tr>
<td>Constraints</td>
<td>Cardinality</td>
</tr>
<tr>
<td>/* constraint methods for the class */</td>
<td>/* is the attribute single or multi-valued */</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>/* class description */</td>
<td></td>
</tr>
</tbody>
</table>
3 Constraints /* constraint methods for the attribute */
Description /* a description of the attribute */

METHOD CLASS
Class Name /* a unique name in all system */
Method Name /* a unique name in the class */
Parameters /* a list of arguments for the method */
Method Type /* the final result data type */
Condition /* the conditional of the method */
Action /* action to be taken for the method */
Description /* the description of the method */

CONSTRAINT CLASS
Class Name /* a unique name in all system */
Method Name /* a unique name in the class */
Parameters /* a list of arguments for the method */
Ownership /* class name of the owner of the method */
Event /* triggered event */
Sequence /* method action time */
Timing /* the method action timer */
Description /* the description of the method */

The header class stores the proper class information and defines the class structure and relationships. Attribute class depicts the properties of a class. It can be filled by values, pointer to other objects, or procedures defined in the method part. It adds procedures for deductive rules functionality to the ordinary EER attribute type [41] and creates a dynamic structure to the Frame model. The Method class depicts rules and behaviour of a class. It denotes the definition of a procedure defined in the attribute part, a deductive rule or an active rule. The Constraint class captures the restrictions on data. It necessitates the knowledge completeness and consistency in the database, and enforces the integrity in the database. Factual data entities are stored in the database through the static class. Rules and Restrictions act on the data invoked by certain event through the active class. Combining these two types of classes within one data model provides the mechanism of structuring and sharing not only data, but also rules and procedures that act on the data to the database.

4 Capture data semantics with semantic metadata for schema translation
The semantic metadata offers the architecture of a metadata for the connectivity and schema translation of heterogeneous databases. Data semantics of different data models can be captured into the semantic metadata, and translated into another data model. A semantic metadata stores an application domain into classes linked via generalization, aggregation and user-defined relationships. Its data are stored in relational tables. The schemas of the source database systems are captured into the semantic metadata, and mapped into a target database in another data model as shown in Figure 1.

Data operation can be used to examine data occurrence of source database which can be interpreted as data semantics. Once captured, the following data semantics can be stored in the metadata trigger procedure:
Step 1.1 Capture the is-a relationship of a legacy database into the semantic metadata
An is-a relationship is a superclass and subclass relationship such that the domain of subclass is a subset of its superclass. The following algorithm examines data occurrence of is-a relationship:
Furthermore, the subclass can inherit attributes from its superclass along with its own attributes in OODB.
Let $tList = \text{a Table List of the Child Table in the Foreign Key List in which all Primary Key attribute is also Foreign Key attribute}$; Begin Search from $tList$

Let $x = \text{Total Record Count of Child Table}$

Let $y = \text{Total Record Count of Parent Table}$

Check has 1:1 relationship in between the parent and child tables; Check has 1:n relationship in between the parent and child tables;

If ($x < y$) AND (Not(is1:1)) AND (Not(is1:n))

Then begin

Parent Table is Super Table,
Child Table is Sub Table,
Parent and child tables are in IS_A relationship

End If

End Search

The following metadata stores the captured is_a relationship:

<table>
<thead>
<tr>
<th>Header Class</th>
<th>Class_Name</th>
<th>Primary_key</th>
<th>Parents</th>
<th>Operation</th>
<th>Class_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_x$</td>
<td>PK($R_x$)</td>
<td>0</td>
<td>$R_x$</td>
<td>Static</td>
<td></td>
</tr>
<tr>
<td>$R_y$</td>
<td>PK($R_y$)</td>
<td>$R_y$</td>
<td>Static</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 1.2 Capture participation semantic of a legacy database into semantic metadata
The existence of a “weak” entity depends on its “strong” entity. Consequently, their relationship is total participation. Otherwise, their relationship is partial participation. The following algorithm examines data occurrence of participation:

<table>
<thead>
<tr>
<th>Relational View</th>
<th>Object-Oriented View</th>
</tr>
</thead>
<tbody>
<tr>
<td>For each 1:n relationship</td>
<td>Given a class: CC, associates to another class AC with its REF(AC), an association attribute of stored OID pointer, we can locate its participation as:</td>
</tr>
<tr>
<td>If the Foreign Key can be nullable</td>
<td>Let participation (CC, AC) = total;</td>
</tr>
<tr>
<td>Then They are in Partial Participation</td>
<td>Select count(*)=C from CC where REF(AC)=null;</td>
</tr>
<tr>
<td>Else They are in Total participation</td>
<td>IF C &gt; 0 THEN participation (CC, AC) = partial;</td>
</tr>
<tr>
<td>End If</td>
<td></td>
</tr>
</tbody>
</table>

Next

The following metadata stores the captured participation:

<table>
<thead>
<tr>
<th>Header Class</th>
<th>Class_Name</th>
<th>Primary_key</th>
<th>Parents</th>
<th>Operation</th>
<th>Class_Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>PK($R$)</td>
<td>0</td>
<td></td>
<td>Call Delete $R$</td>
<td>Active</td>
</tr>
<tr>
<td>$R_j$</td>
<td>PK($R_j$)</td>
<td>0</td>
<td></td>
<td>Call Create $R_j$</td>
<td>Active</td>
</tr>
</tbody>
</table>

Method class

<table>
<thead>
<tr>
<th>Method_ name</th>
<th>Class name</th>
<th>Para meter</th>
<th>Seq no</th>
<th>Method type</th>
<th>Condition</th>
<th>Action</th>
<th>Next Seq no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete $R$</td>
<td>$R$</td>
<td>@PK($R$)</td>
<td></td>
<td></td>
<td>If (Select * from $R_j$ where PK($R_j$) = @PK($R$) ≠ null)</td>
<td>Delete * from $R_j$ where PK($R_j$) = @PK($R$)</td>
<td></td>
</tr>
<tr>
<td>Create $R_j$</td>
<td>$R_j$</td>
<td>@PK($R_j$)</td>
<td></td>
<td></td>
<td>If (Select * from $R$ where PK($R$) = @PK($R_j$) ≠ null)</td>
<td>Insert (@PK($R_j$))</td>
<td>$R_j$</td>
</tr>
</tbody>
</table>

Step 1.3 Capture generalization of a legacy database schema into semantic metadata
A generalization can be represented by more than one subclasses having a common superclass. The following algorithm examines data occurrence of disjoint generalization such that subclass instances are mutually exclusive stored in each subclass.

Relational View

Object-Oriented View

Let tList a = Super Table List from IS_A relationship List such that the Super Table has two or more Sub Table; For Each Super Table in tList Let subTableList = the corresponding Sub Table; Calculate the duplicate Record Count from each two Sub Table If there are duplicate Record between Sub Tables Then Super Table and sub tables are in Overlap relationship Else Super Table and sub tables are in Disjoint relationship End If

Let a superclass and its OIDs: C, OID(C), referring to its subclass and their OID: C j1, OID(C j1), … C jn, OID(C jn), their generalization can be located as:

If IS_A-relationship (C j1, C) = True and … and IS_A-relationship (C jn, C) = True Then Generalization (C, C j1, … C jn) := Disjoint;

For h := 1 to n do 
  For k := 1 to n do 
    IF h < k
      THEN begin 
        Select Count(OID(C ih)) from C ih; 
        Select Count(OID(C ik)) from C ik;
        Join_OID = OID(C ih) Union OID(C ik)
        Join_Count = select count (Join_OID)
        IF Join_Count < Count(OID(C ih)) + Count(OID(C ik)) THEN Begin 
          Generalization (C, C j1, …, C jn) := Overlap;
          Exit;
        End;
      End;
    Next; /* for loop k */
  Next; /* for loop h */

The following metadata stores the captured disjoint generalization:

<table>
<thead>
<tr>
<th>Header Class</th>
<th>Operation</th>
<th>Class_Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Call Create R j1</td>
<td>Active</td>
</tr>
<tr>
<td>R2</td>
<td>Call Create R j2</td>
<td>Active</td>
</tr>
</tbody>
</table>

Method class

<table>
<thead>
<tr>
<th>Method_Name</th>
<th>Class_name</th>
<th>Param Parameter</th>
<th>Seq no</th>
<th>Method_Type</th>
<th>Condition</th>
<th>Action</th>
<th>Next Seq no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create_R j1</td>
<td>R j1</td>
<td>@ PK(R j1)</td>
<td></td>
<td>Boolean</td>
<td>If (Select * from R j1 where PK(R j1) = @ PK(R j1)) = true</td>
<td>Create_R j1 = true</td>
<td></td>
</tr>
<tr>
<td>Create_R j2</td>
<td>R j2</td>
<td>@ PK(R j2)</td>
<td></td>
<td>Boolean</td>
<td>If (Select * from R j2 where PK(R j2) = @ PK(R j2)) = true</td>
<td>Create_R j2 = true</td>
<td></td>
</tr>
</tbody>
</table>

Step 1.4 Capture cardinality of schema in a legacy database into the semantic metadata

The cardinality specifies data volume relationship in the database. The following algorithm examines data occurrence of cardinality of 1:1, 1:n and n:m.

Relational View

Object Oriented View

Discover 1:1
Let x = the Total Record Count of Child Table
Let y = the Total Record Count of Parent Table
Let z = the Distinct Record Count of Child Table
If (x = y) AND (y = z) Then they are 1:1

Discover 1:n for cases
Let x = the Total Record Count of Child Table
Let y = the Total Record Count of Parent Table
Let z = the Distinct Record Count of Child Table
Check if there is null value in Child Table = n
Case 1: (z < y)
Case 2: ((z = y + 1) AND (n))
Case 3: (z = y) AND (z > y)
Case 4: (z = y) AND (z < y) AND (n)

Given two classes and their reference attributes C1, Ref(C1), C2, Ref(C2) in an OO schema S, we can locate its cardinality as:

For i = 1 to 2
  Select C i, Ref(C i), from S;
  max(i) = 1
While not end of instance(Ref(C i)) do
  Begin If instance(Ref(C i)) = NULL
    Minimum = True;
    Else If Ref(C i) is a set reference
      then max(i) = n;
  End;
End;
If Minimum
Discover m:n
Get a Table List of the Child Table in 1:n relationship
which has two or more 1:n relationship = tList
If (Primary Key of Child Table in tList is
  compound Primary Key) and (All Primary Keys
  are also the Foreign Key in such relationship)
then It is m:n relationship
End If
End If

The following metadata stores the captured 1:n cardinality between R and Rj:

<table>
<thead>
<tr>
<th>Attribute Class</th>
<th>Class_name</th>
<th>Attribute_Name</th>
<th>Method_Name</th>
<th>Attribute_type</th>
<th>Default_value</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Rj</td>
<td></td>
<td></td>
<td></td>
<td>n</td>
<td></td>
<td>Associated class attribute</td>
</tr>
<tr>
<td>Rj</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>Associated class attribute</td>
</tr>
</tbody>
</table>

Step 1.5 Capture aggregation of a legacy database schema into the semantic metadata.
Aggregation is an abstraction concept for building composite objects from their component objects.
The following algorithm examines data occurrence of aggregation object which must consist of all of
its component objects: (note: Reference attribute refer to the component class)

Relational View

<table>
<thead>
<tr>
<th>Class_Name</th>
<th>Attribute_Name</th>
<th>Method_Name</th>
<th>Attribute_type</th>
<th>Default_value</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR1</td>
<td>PK(CR1)</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>static</td>
</tr>
<tr>
<td>CR2</td>
<td>PK(CR2)</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>static</td>
</tr>
<tr>
<td>AR</td>
<td>PK(CR1), PK(CR2)</td>
<td></td>
<td></td>
<td>0</td>
<td>Call Create_AR</td>
<td>active</td>
</tr>
</tbody>
</table>

Method class

<table>
<thead>
<tr>
<th>Method_Name</th>
<th>Class_name</th>
<th>Parameter</th>
<th>Seq_no</th>
<th>Method_type</th>
<th>Condition</th>
<th>Action</th>
<th>Next_seq_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create_AR</td>
<td>AR</td>
<td>@PK(CR1), @PK(CR2)</td>
<td></td>
<td></td>
<td>If ((Select * from CR1 where PK(CR1) = @PK(CR1)) ≠ null) and If ((Select * from CR2 where PK(CR2) = @PK(CR2) ≠ null)</td>
<td>Insert AR (@PK(CR1), @PK(CR2))</td>
<td></td>
</tr>
</tbody>
</table>

Translating the semantic metadata to a legacy database schema
Step 1.2 Translating the semantic metadata to a legacy database schema
If the Target database is an OODB, the object oriented data model holds the method and constraint class from the semantic metadata. If the Target database is a RDB, the method class from the semantic metadata is implemented using stored procedures; and the constraint class is implemented using triggers as shown below:

- Step 1.2.1 Map semantic metadata to relational schema
  For i = 1 to M do
  begin
  Get objects \((\text{FMO})_i\) in Semantic metadata \((\text{FM})\);
  Create table \(T_i\) using \((\text{FMO})_i\);
  end;
- Step 1.2.2 Map semantic metadata to object-oriented schema
  For i = 1 to M do
  begin
  Get objects \((\text{FMO})_i\) in Frame Model \((\text{FM})\);
  Create class \(C_i\) using \((\text{FMO})_i\);
  end;

5. Concurrent Data Materialization

One set of data produced from the source RDB or OODB can be materialized into another target RDB or OODB depending on user needs. The result is an interchangeable data set between RDB and OODB as shown in figure 2:

Step 2.1: Capture Data from the Source Database into the ASCII Files
The captured data will be output to sequential files. If the Source database is a RDB, the ASCII file formed contains the key value of this tuple and the attributes with base type. The Object_Key attribute contains the key value and the Attribute1_Name, … AttributeN_name contain the attributes of base type. For each class, one ASCII file will be formed. If the Source database is an OODB, the ASCII file formed contains the OID (called ObjKey_Value here) of this object, the attributes with base type, the association attributes with the OID of objects within associated class (called AssAtr1Key_Value…
here), and the set value type attributes that contain attributes of the base type (called SetAtr1_Value...) as shown in table 2.

**Table 2 File Structure for an ASCII file stores captured data from the Source database**

<table>
<thead>
<tr>
<th>ASCII File Name: Class_Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Object_Key</td>
<td>ObjKey_Value</td>
</tr>
<tr>
<td>Attribute1_Name</td>
<td>Attribute1_Value</td>
</tr>
<tr>
<td>Attribute2_Name</td>
<td>Attribute2_Value</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>AttributeN_Name</td>
<td>AttributeN_Value</td>
</tr>
<tr>
<td>AssocAtr1_Name</td>
<td>AssocAtr1Key_Value</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>SetAtr1_Name</td>
<td>SetAtr1_Value1</td>
</tr>
<tr>
<td>SetAtr1_Value2</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Step 2.1.1: Capture Data from the Relational Database to the ASCII Files**
The data will be downloaded into the ASCII files directly from the RDB.

**Step 2.1.2: Data materialization from the OODB to the ASCII Files**
This step captures the OID and data of the base types attributes of every object within the chosen class to the sequential files. For each object in the class, the values of the association attributes will be captured to the sequential files. For each object in the class, the values of the set values attributes are captured into sequential files as follows:

Given: input: classes C_i to C_p within the target OODB
- objects O_{ij} to O_{ijQ} within the class C_i
- classes (AC)_{ik} with association attributes A_{ik} (from A_{i1} to A_{iR}) within Class C_i
- set values attributes S_{i1} to S_{iT} in class C_i

output: OID and data of the base types attributes to sequential file (SF)_i

For i = 1 to P do
  begin
    for j = 1 to Q do
      begin
        Get OID and data of all attributes from objects O_{ij};
        Output OID and data of the base types attributes to sequential file (SF)_i;
      end;
    For k = 1 to R do
    begin
      Case association of
      One-to-one association with A_{ik}
        Get OID from (ACO)_{ijk} and output to the sequential file (SF)_i;
      One-to-many association with A_{ik}
        Get all OIDs from (AC)_{ik} and output to (SF)_i;
      end;
    For m = 1 to T do
    begin
      If S_{im} within C_i is of base types
      then Output value of object O_{ijm} that own S_{im} attribute to the sequential file (SF)_i;
      else Get all OIDs from (SC)_{im} and output to (SF)_i;
    end;
  end;

**Step 3: Data materialization from the ASCII Files into the Target Database**
The process will read object data from the ASCII file according to the semantic in the semantic metadata and materializes data into the Target database.

**Step 3.1: Data materialization from the ASCII files to the RDB**
After reading data in the sequential files, data is materialized into the target relational database. The set values attributes’ values are materialized into other tables in the target relational database as follows:

Given input: single value (SD)_{ij} in Sequential file S_i
- set value ((SV)_{ijk})_{im} in Sequential file S_i
output: tables T_1 to T_M within the target Relational database

For i = 1 to M do
For \( j = 1 \) to \( N \) do
Get data \((SD)_j\) containing all attributes’ values of the record \( R_j \) from the sequential file \( S_i \);
Insert record \( R_j \) by using \((SD)_j\);
If Set values attributes exist for \( R_j \)
then begin
For \( k = 1 \) to \( P \) do
begin
For \( m = 1 \) to \( Q \) do
begin
Get set value \( ((SV)_{ijk})_m \);
Insert \((OKV)_j\) value and \( ((SV)_{ijk})_m \) value into record \( ((RS)_{ijk})_m \) in table \((TS)_{ik} \);
end;
end;
end;
end;
end;
end;

**Step 3.2: Data Materialization from the ASCII files to the OODB**

After reading data in the sequential files, the data with non-associated key attributes is materialized to the target OODB. Non-associated key attributes with OIDs within each class are generated. The OID of every object in each subclass is updated as the OID of the same object in the superclass. The value of association attributes will be updated as the OID of their associated object in other class. The set values of objects are updated to hold a set of associated OIDs (i.e. stored OID) as follows:

Given: single value \((SD)_j\) in Sequential file \( S_i \)
set value \( ((SV)_{ijk})_m \) in Sequential file \( S_i \)
output: Classes \( C_1 \) to \( C_M \) within the target Object-oriented database

For \( i = 1 \) to \( M \) do
begin
For \( j = 1 \) to \( N \) do
Get data \((SD)_j\) with all non-associated key attributes’ values of object \( O_j \) from sequential file \( S_i \);
Insert object \( O_j \) by using \((SD)_j\);
end;
end;

For \( i = 1 \) to \( M \) do
begin
For \( j = 1 \) to \( N \) do
Get data \((SD)_j\) with all non-associated key attributes’ values of object \( O_j \) from sequential file \( S_i \);
Get object key \((OK)_j\) with value \((OKV)_j\) from the sequential file \( S_i \);
Get object \( O_j \) in Class \( C_i \);
Get superclass and subclass relationship from the Frame model (FM);
If Class \( C_i \) has superclass (CS)
then begin
Get OID \((POID)_{ij}\) of superclass object \( (SO)_{ij} \) in (CS) by using the values \( (OKV)_j \);
Update the OID in object \( O_j \) with \((POID)_j\);
end;
end;
end;

For \( i = 1 \) to \( M \) do
begin
For \( j = 1 \) to \( N \) do
Get data \((SD)_j\) with all non-associated key attributes’ values of object \( O_j \) from sequential file \( S_i \);
Get object \( O_j \) in Class \( C_i \);
If \( O_j \) has association attributes contained in Si
then begin
For \( k = 1 \) to \( P \) do
Case association of
One-to-one association with object \( O_j \):
begin
Get association attribute \((AA)_{ijk}\) with value \((AAV)_{ijk}\) from the sequential file \( S_i \);
Get OID \((AOID)_{ijk}\) of the associated object \( (AO)_{ijk} \) using the values \((AAV)_{ijk}\);
Update blank association attribute \((AA)_{ijk}\) in object \( O_j \) with \((AOID)_{ijk}\);
end;
One-to-many association with object Oij:
begin
For m = 1 to Q
Get association attribute (AA)_{ijkm} with value (AAV)_{ijkm} from sequential file S_i;
Get OID (AOID)_{ijkm} of associated object (AO)_{ijkm} using values (AAV)_{ijkm};
Update association attribute (AA)_{ijk} in object Oij with added (AOID)_{ijkm};
end;
end;
end;
end;
end;
end;
end;
end;
end;

For i = 1 to M do
begin
For j = 1 to N do
Get data (SD)_{ij} with all non-associated key attributes' values of object Oij from sequential file S_i;
Get object Oij in Class C_i;
If Oij has some set values attributes contained in S_i then begin
For k = 1 to P do
For m = 1 to Q do
Get set value ((SV)_{ijk})_m from sequential file S_i;
If set value ((SV)_{ijk})_m is of base types
then Update set values attribute (SA)_{ijk} in object Oij with added ((SV)_{ijk})_m;
else begin
Get OID (SOID)_{ijkm} of set values object (SO)_{ijkm} in other class using values ((SV)_{ijk})_m;
Update set values attribute (SA)_{ijk} in object Oij with added (SOID)_{ijkm};
end;
end;
end;
end;
end;
end;

In summary, a database can be materialized without any loss of information if p maps a state of a relational database into an object-oriented database and p' maps a state of an object-oriented database into a relational database, then it can be shown that \( p(p'( R )) = R \) where R is the relational database before materialization.

6 Verification of data materialization rules

Information Capacity equivalence [42] is a formal approach taken by us to provide sets of equivalence preserving transformations of the schema and data materialization without information loss. A classification of generic integration and translation tasks based on their operation goals is defined by Rosenthal and Reiner. There are different levels of operational goals (G1, G2, G3, G4) in using database systems. They range from queries to update. We prove that the state of the database is the same for both the source schema (Object oriented database schema) and the target schema (Relational database schema). It is valuable to analyse whether the original schema and the transformed schema after the mapping function are in a dominance or equivalence domain. The mapping function to extend the information capacity is required to provide different levels of goals. Mapping functions can be classified into five categories that have different information capacity.

1. Functional
Let A and B be sets. The mapping is Functional if \( f : A \to B, \forall a \in A, \exists b \in B \cdot f(a) = b \)

2. Injective
If the inverse of the binary mapping relation is also Functional, the function is injective. \( f^{-1} : B \to A, \forall b \in B, \exists a \in A \cdot f^{-1}(b) = a \)

3. Total
If the Functional binary mapping relation is defined on every element of \(A\), then the function is total. It is an information capacity preserving mapping between the instance of two schemas \(S1\) and \(S2\). (\(S1\) denotes the original schema and \(S2\) denotes the transformed schema.)

\[ f : I(S1) \rightarrow I(S2) \]

where \(I(Sn)\) denotes the set of all (data) instances of schema \(Sn\) and \(S1\) dominates \(S2\) (i.e. \(S2 \leq S1\)), that is \(\forall I(S1) \in S1, \exists I(S2) \in S2\)

\[ f^{-1} : I(S2) \rightarrow I(S1) \]

4. Surjective

If the inverse of the function is total, it is an information capacity preserving mapping between the (data) instance of two schema \(S1\) and \(S2\). This Total and injective function is called a Surjective function.

\[ f^{-1} : I(S2) \rightarrow I(S1) \text{ and } S2 \text{ dominates } S1 \text{ (i.e. } S1 \leq S2) \text{, that is } \forall I(S2) \in S1, \exists I(S1) \in S1 \]

5. Bijection

If the mapping function meets all the above properties, it is an information equivalence preserving mapping.

\[ f : I(S1) \rightarrow I(S2) \]

where \(I(Sn)\) denotes the set of all (data) instances of schema \(Sn\) and \(S1\) and \(S2\) are equivalent via \(f\).

That is \(\forall I(S1) \in S1, \exists I(S2) \in S2 \wedge \forall I(S2) \in S2, \exists I(S1) \in S1 \bullet S1 \equiv S2\)

The four levels of operational Goals (G1, G2, G3 and G4) for systems involving two schemas and their relative information capacity are:

- **G1** targets to make a query via \(S1\) the data stored under \(S2\), where \(S1\) is a view of \(S2\). In other words, \(S1\) provides a logical view on the physical database of \(S2\). Therefore, a Total function is required at instance level for achieving this minimum operational goal. As the query \(q\) on \(I(S1)\) is mapped to the unique query \(q\) on \(I(S2)\), the function does not have to be information preserving to achieve G1. The information capacities of \(S1\) and \(S2\) may be incommensurate.

- **G2** can achieve G1. Moreover it can also be used to view through \(S1\) the entire database stored under \(S2\). A Total Injective function is required to achieve G2 because the function needs no loss of information where information preserving mapping is required to achieve G2. \(S1\) must dominate \(S2\).

- **G3** can achieve G2. Moreover it can also be used to update the data stored under \(S2\) using the view \(S1\). A Total Injective and Surjective function is required. An update \(u\) that changes instance of \(S1\), \(i1\) to a new instance \(i1'\), that is \(u(i1) = i1'\). Instance \(i1'\) should determine a unique instance of \(S2\). As a result, \(f\) must be Surjective. As the function \(f\) must be an information preserving mapping to achieve G3, \(S2\) must dominate \(S1\).

- **G4** targets to query through \(S1\) that data stored under \(S2\) and also through \(S2\) the data stored under \(S1\). A bi-direction and information preserving mapping in both direction is needed. \(S1\) and \(S2\) must be equivalent in both directions to allow updates be done through both \(S1\) and \(S2\). Function \(f\) is a Bijection function to achieve G4, \(S1\) and \(S2\) must be equivalent.

In our approach, a successful data materialization should require information capacity of the original schemas to be equivalent or dominated by the transformed schema. In order to achieve the information capacity needs, proof will be shown that each proposed data conversion process can fulfill up to the third level of its operational goal (G3) to ensure information completeness. The following three major steps must be preformed.

**Step 1. Converting One-to-many association (Set Attributes)**

Example:

![Diagram of Schema A and Schema X](image-url)
Rule
Let \( C, D, E \) be the class \( A \) (where class \( A \) contains set attributes), table \( X_1 \) and table \( X_2 \) (where \( X_2 \) is a generated table containing set attribute values) respectively. Let \( \text{SetAtt} \) be a function to select the attribute type from the Class \( C \) attribute that is of set attribute type. IF class \( A \) contains set attributes THEN begin
\[
\text{dom}(D) \leftarrow \text{dom}(C) - \text{SetAtt}(C) \quad : \quad E \leftarrow \text{Pr iKey}(C) \cup \text{SetAtt}(C) \\
\text{Cardinality}(D, E) \leftarrow 1:n \\
\forall e \in E, \exists d \in D \bullet \text{ForKey}(e) = \text{Pr iKey}(d)
\]
end

Pre-cond.
\( \text{SetAtt}(C) \neq \emptyset \)

Post-Cond
\[
\text{dom}(D) \leftarrow \text{dom}(C) - \text{SetAtt}(C) \quad : \quad E \leftarrow \text{Pr iKey}(C) \cup \text{SetAtt}(C) \\
\text{Cardinality}(D, E) \leftarrow 1:n \\
\forall e \in E, \exists d \in D \bullet \text{ForKey}(e) = \text{Pr iKey}(d)
\]

Proof:
Let \( f : (C) \rightarrow (D, E) \) be the step to map class \( A \) to table \( X_1 \) and \( X_2 \)

**Proof of \( f \) is functional**
\( (D, E) \) provides a view to \( C \). From the mapping rule, tuples in \( D \) and \( E \) can be found such that \( D \) has all the attribute type as \( C \) except the set attribute that formed by the function \( \text{SetAtt} \). \( E \) contains the primary key \( C \) and the values of the set attribute type.
\[
\forall c \in C \land \text{SetAtt}(C) \neq \emptyset, \exists d \in D, e \in E \land \text{dom}(d) = \text{dom}(c) \land \text{SetAtt}(c) \land e = \text{Pr iKey}(c) \cup \text{SetAtt}(c) \land \text{RD}(d, e) \bullet \text{ForKey}(e) = \text{Pr iKey}(d)
\]
Hence \( f \) is functional

2. **Proof of \( f \) is total**
Since \( f \) is defined for every elements of \((C, D)\), \( f \) is total.

3. **Proof of \( f \) is Injective**
\((C, D)\) are mapped into \((E, F)\) and preserved in \((C, D)\). There is a bi-directional one to one mapping between elements of \((C, D)\) and \((E, F)\).
\[
\forall (e,f) \land (E,F) \in \text{RD}(e,f) \land e = \text{ForKey}(f) = \text{PriKey}(e), \exists c \in C, d \in C \land \text{RC}(c,d) \bullet f^{-1}(e,f) = (c, d)
\]
For every element in \((E, F)\), the corresponding element in \((C, D)\) can be found. Hence \( f \) is injective

4. **Proof of \( f \) is Surjective**
Since \( f^{-1}\) is defined for every elements of \((E, F)\), \( f^{-1}\) is total. Hence \( f \) is surjective

5. **Proof of \( f \) is a Bijection**
From Proof 1, 2, 3 and 4, \( f \) is a bijection. \((E, F)\) is proved to be equivalent to \((C, D)\)

Step 2. Converting One-to-many association

Example:
Rule

Let $C, D, E, F$ be the class A, class B (where class A is on the “one” side and class B is on the “many” side), table X1 and table X2 (where class X1 is on the “one” side and class X2 is on the “many” side) respectively.

IF class A has a one to many relationship with class B

THEN begin

\[ E \leftarrow C : F \leftarrow D : \text{Cardinality}(E, F) \leftarrow 1 : n \]

\[ \forall f \in F, \exists e \in E \bullet \text{ForKey}(f) = \text{PriKey}(e) \]

end

Pre-cond.

Association $(C, D) \leftarrow 1 : n : \exists e \in C, d \in D \bullet RC(c, d)$ and RC is a one to many association.

Post-Cond

\[ E \leftarrow C : F \leftarrow D : \text{Cardinality}(E, F) \leftarrow 1 : n \]

\[ \exists e \in E, f \in F \bullet RD(e, f) \] and RD is a one to many relation

\[ \forall f \in F, \exists e \in E \bullet \text{ForKey}(f) = \text{PriKey}(e) \]

Proof:

<table>
<thead>
<tr>
<th>Step</th>
<th>Proof of $f$ is functional</th>
</tr>
</thead>
</table>
| G1     | $(E, F)$ provides a view to $(C, D)$. From the mapping rule, tuples in E and F can be found that tuples in E has a one to many association with tuples in F, and tuples in E has the same attributes as class C, and tuples in F has the same attributes as class D.

\[ \forall (c, d) \in (C, D) \land RC(c, d), \exists e \in E, f \in F \land RD(e, f) \land \text{ForKey}(f) = \text{PriKey}(e) \]

Hence $f$ is functional |

2. Proof of $f$ is total

Since $f$ is defined for every elements of $(C, D), f$ is total.

3. Proof of $f$ is Injective

$(C, D)$ are mapped into $(E, F)$ and are preserved in $(C, D)$. There is a bi-directional one to one mapping between elements of $(C, D)$ and $(E, F)$.

\[ \forall (e, f) \in (E, F) \land RD(e, f) \land e = \text{ForKey}(f) = \text{PriKey}(e), \exists c \in C, d \in C \land RC(c, d) \]

\[ \bullet f^{-1}(e, f) = (c, d) \]

For every element in $(E, F)$, the corresponding element in $(C, D)$ can be found. Hence $f^{-1}$ is injective |

G2

4. Proof of $f$ is Surjective

Since $f^{-1}$ is defined for every elements of $(E, F), f^{-1}$ is total. Hence $f$ is surjective

5. Proof of $f$ is a Bijection

From Proof 1, 2, 3 and 4, $f$ is a bijection. $(E, F)$ is proved to be equivalent to $(C, D)$

5. Proof of $f$ is a Bijection

From Proof 1, 2, 3 and 4, $f$ is a bijection. $(E, F)$ is proved to be equivalent to $(C, D)$

Step 3 Converting many-to-many association
Example:

Let $C, D, E, F, G$ be the class A, class B, table X1 and table X2 and X3 (containing primary key of X1 and X2) respectively.

**IF** class A has a many to many association with class B

**THEN** begin

$$E \leftarrow C : F \leftarrow D : G \leftarrow \text{Pr}_i\text{Key}(E) \cup \text{Pr}_i\text{Key}(F)$$

Cardinality $(E, G) \leftarrow 1:n : \forall g \in G, \exists e \in E \bullet \text{ForKey}(g) = \text{Pr}_i\text{Key}(e)$

Cardinality $(F, G) \leftarrow 1:m : \forall g \in G, \exists f \in F \bullet \text{ForKey}(g) = \text{Pr}_i\text{Key}(f)$

end

**Pre-cond.** Cardinality $(C, D) \leftarrow n : m : \exists c \in C, d \in D \bullet \text{RC}(c, d)$ and RC is a many to many relation

**Post-Cond.**

$$E \leftarrow C : F \leftarrow D : \text{Cardinality}(E, F) \leftarrow 1:n$$

$$\exists e \in E, g \in G \bullet \text{RD}(e, g)$$ and RD is a one to many relationship

$$\forall g \in G, \exists e \in E \bullet \text{ForKey}(g) = \text{Pr}_i\text{Key}(e)$$

$$\exists f \in E, g \in G \bullet \text{RE}(f, g)$$ and RE is a one to many relationship

$$\forall g \in G, \exists f \in F \bullet \text{ForKey}(g) = \text{Pr}_i\text{Key}(f)$$

**Proof:**

Let $f : (C, D) \rightarrow (E, F, G)$ be the step to map class A and B to table X1, X2 and X3

**G1**

**Proof of $f$ is functional**

$(E, F, G)$ provides a view to $(C, D)$. From the mapping rule, tuples in E, F, G can be found that tuples in E has a one to many association with tuples in G, tuples in F has a one to many association with tuples in G, tuples in E has the same attributes as class C, and tuples in F has the same attributes as class D.

$\forall (c, d) \in (C, D) \land \text{RC}(c, d), \exists (e, f, g) \in (E, F, G) \land \text{RD}(e, g) \land \text{RE}(f, g) \land \text{ForKey}(g) = \text{Pr}_i\text{Key}(f) \land \text{ForKey}(g) = \text{Pr}_i\text{Key}(f) \land f(c, d) = (e, f, g)$

Hence $f$ is functional.

**2. Proof of $f$ is total**

Since $f$ is defined for every elements of $(C, D)$, $f$ is total.

**G2**

**3. Proof of $f$ is Injective**

$(C, D)$ are mapped into $(E, F, G)$ and preserved in $(C, D)$. There is a bi-directional one to one mapping between elements of $(C, D)$ and $(E, F, G)$.

$\forall (e, f, g) \in (E, F, G) \land \text{RD}(e, g) \land \text{RE}(f, g) \land \text{ForKey}(g) = \text{Pr}_i\text{Key}(f) \land \text{ForKey}(g) = \text{Pr}_i\text{Key}(f), \exists c \in C, d \in C \land \text{RC}(c, d) \land f^{-1}(e, f, g) = (c, d)$

For every element in $(E, F, G)$, the corresponding element in $(C, D)$ can be found.

Hence $f$ is injective.

**G3**

**4. Proof of $f$ is Surjective**

Since $f^{-1}$ is defined for every elements of $(E, F, G)$, $f^{-1}$ is total, Hence $f$ is surjective.

**G4**

**5. Proof of $f$ is a Bijection**

From Proof 1, 2, 3 and 4, $f$ is a bijection. $(E, F, G)$ is proved to be equivalent to $(C, D)$.
Step 4. Converting Inheritance

Example:

<table>
<thead>
<tr>
<th>Class A1</th>
<th>Person</th>
<th>Schema B1</th>
<th>Table B1</th>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Primary Key: Person_OID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class A2</th>
<th>Student</th>
<th>Schema B2</th>
<th>Table B2</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Primary Key: Person_OID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class A3</th>
<th>FT_Student</th>
<th>Schema B3</th>
<th>Table B3</th>
<th>FT_Student</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Primary Key: Person_OID</td>
</tr>
</tbody>
</table>

| Rule | Let $A_1, A_2, A_3 \cdots A_n$ be classes ( $A_n$ “is-a” $A_{n-1} \cdots A_2$ “is-a” $A_1$ ) and tables $B_1, B_2, B_3 \cdots B_n$ are mapped from them and OID of $A_i$ be $A_i$ .oid . IF $A_1, A_2, A_3 \cdots A_n$ classes ( $A_n$ “is-a” $A_{n-1} \cdots A_2$ “is-a” $A_1$ ) THEN begin $B_1 \leftarrow A_i \cdots B_n \leftarrow A_n : \text{Pr iKey}(B_i) = \cdots \text{Pr iKey}(B_n) = A_i$ .oid end |

| Pre-cond. | $A_1, A_2, A_3 \cdots A_n$ classes ( $A_n$ “is-a” $A_{n-1} \cdots A_2$ “is-a” $A_1$ ) |

| Post-Cod | $B_1 \leftarrow A_i \cdots B_n \leftarrow A_n : \text{Pr iKey}(B_i) = \cdots \text{Pr iKey}(B_n) = A_i$ .oid |

| Proof: | Let $f_i : A_i \rightarrow B_i$ be a set of steps to map classes $A_1, A_2, A_3 \cdots A_n$ to tables $B_1, B_2, B_3 \cdots B_n$. |

G1 | Proof of $f_i$ is functional $B_i$ provides a view to $A_i$. From the mapping rule, table $B_i$ can be found such that it has the primary key $A_i$ oid and the same attributes as class $A_i$. $\forall a_i \in A_i, \exists b_i \in B_i \land \text{Pr iKey}(b_i) = A_i$ oid $\bullet f_i(a_i) = b_i$ Hence $f_i$ is functional |

G2 | 2. Proof of $f_i$ is total Since $f_i$ is defined for every elements of $A_i$, $f_i$ is total. |

3. Proof of $f_i$ is Injective $A_i$ is mapped into $B_i$ and preserved in $A_i$. There is a bi-directional one to one mapping between elements of $A_i$ and $B_i$. $\forall b_i \in B_i \land \text{Pr iKey}(b_i) = A_i$ oid, $\exists a_i \in A_i \cdot f_i^{-1}(b_i) = a_i$ For every element in $B_i$, the corresponding element in $A_i$ can be found. Hence $f_i$ is injective |

G3 | 4. Proof of $f_i$ is Surjective |
Since \( f_i^{-1} \) is defined for every elements of \( B_i \), \( f_i^{-1} \) is total. Hence \( f_i \) is surjective.

**5. Proof of \( f_i \) is a Bijection**

From Proof 1, 2, 3 and 4, \( f_i \) is a bijection.

**6. Proof of \( (A_1, A_2, A_3 \cdots A_n) \) is equivalent \( B_1, B_2, B_3 \cdots B_n \)**

From Proof 5, all \( f_i \) are bijections, hence \( (A_1, A_2, A_3 \cdots A_n) \) is equivalent \( B_1, B_2, B_3 \cdots B_n \)

7 Prototypes

The flow chart for the schema translation from the RDB to the semantic metadata is shown in figure 3. It examines data occurrence to capture advanced semantics in the RDB. A semantic knowledge base is required as an intermediate storage for all primitive semantics of the is_a and cardinality relationship. When the data examination process is completed with the semantics were extracted, the captured semantics are translated into the semantic metadata.

![Flow chart for Schema translation from a RDB to the Semantic metadata](image)

Figure 3 Flow chart for Schema translation from a RDB to the Semantic metadata

The data flow diagram of the schema translation from the semantic metadata to the OODB is shown in figure 4. The Header and Attribute classes in the frame model contain class entities and attribute data types. They form the basic class structure in the OODB. Advanced semantics are translated into the methods in the OODB schema through the Event Extraction Process and Method Extraction Process.
Case study: Materializing RDB to OODB:

Relation Patient  
Relation Borrower  
Relation Borrow  
Relation Medical_rec  
Relation Ward_rec  
Relation Outpatient_rec  
Relation AE_record  
Relation Department  
Relation Doctor  
Relation Other-hospital  
Relation Record_folder  
Relation Loan_history  
Relation Insurance_cover

The whole database resides in the Microsoft SQL Server 7.0. Notice that underline means primary key and * means foreign key. The relational schema is listed below:

Relation Patient: (HKID, Patient_name)
Relation Borrower: (*Borrower_no, Borrower_name)
Relation Borrow: (*Borrower_no, *Folder_no)
Relation Medical_rec: (Medical_rec_no, Create_date, Sub_type, *Folder_no)
Relation Ward_rec: (*Medical_rec_no, Ward_no, Admission_date, Discharge_date)
Relation Outpatient_rec: (*Medical_rec_no, OPD_no, Specialty)
Relation AE_record: (*Medical_rec_no, AE_no)
Relation Department: (Borrower_no, Department_name)
Relation Doctor: (Borrower_no, Doctor_name)
Relation Other-hospital: (Borrower_no, Hospital_name)
Relation Record_folder: (Folder_no, location, *HKID)
Relation Loan_history: (*Borrower_no, *Folder_no, Loan_date)
Relation Insurance_cover: (Insurance_nm, *HKID)

The data contained in the database is as follows:

Table Doctor

<table>
<thead>
<tr>
<th>Borrower_no</th>
<th>Doctor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bloor</td>
</tr>
<tr>
<td>2</td>
<td>Smith</td>
</tr>
<tr>
<td>3</td>
<td>Kim</td>
</tr>
<tr>
<td>4</td>
<td>Chitson</td>
</tr>
<tr>
<td>5</td>
<td>Navathe</td>
</tr>
</tbody>
</table>

Table Department

<table>
<thead>
<tr>
<th>Borrower_no</th>
<th>Department_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>X-Ray</td>
</tr>
<tr>
<td>12</td>
<td>Infant</td>
</tr>
<tr>
<td>13</td>
<td>Chest</td>
</tr>
<tr>
<td>14</td>
<td>Skin</td>
</tr>
<tr>
<td>15</td>
<td>Therapy</td>
</tr>
</tbody>
</table>

Table Borrower

<table>
<thead>
<tr>
<th>Borrower_no</th>
<th>Borrower_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bloor</td>
</tr>
<tr>
<td>2</td>
<td>Smith</td>
</tr>
<tr>
<td>3</td>
<td>Kim</td>
</tr>
<tr>
<td>11</td>
<td>X-Ray</td>
</tr>
<tr>
<td>12</td>
<td>Infant</td>
</tr>
<tr>
<td>14</td>
<td>Skin</td>
</tr>
<tr>
<td>21</td>
<td>Mac Neal</td>
</tr>
<tr>
<td>22</td>
<td>Riveredge</td>
</tr>
<tr>
<td>25</td>
<td>Golden Park</td>
</tr>
</tbody>
</table>
**Table Record_folder**

<table>
<thead>
<tr>
<th>Folder_no</th>
<th>Location</th>
<th>HKID</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_21</td>
<td>Hong Kong</td>
<td>E3766849</td>
</tr>
<tr>
<td>F_22</td>
<td>Kowloon</td>
<td>E8018229</td>
</tr>
<tr>
<td>F_23</td>
<td>New Territories</td>
<td>E6077888</td>
</tr>
</tbody>
</table>

**Table Insurance_cover**

<table>
<thead>
<tr>
<th>Insurance_no</th>
<th>HKID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E3766849</td>
</tr>
<tr>
<td>2</td>
<td>E8018229</td>
</tr>
</tbody>
</table>

**Table Patient**

<table>
<thead>
<tr>
<th>HKID</th>
<th>Patient name</th>
</tr>
</thead>
<tbody>
<tr>
<td>E3766849</td>
<td>Smith</td>
</tr>
<tr>
<td>E8018229</td>
<td>Bloor</td>
</tr>
<tr>
<td>E6077888</td>
<td>Kim</td>
</tr>
</tbody>
</table>

**Table Loan_history**

<table>
<thead>
<tr>
<th>Borrower_no</th>
<th>Folder_no</th>
<th>Loan_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F_21</td>
<td>Jan-10-2000</td>
</tr>
<tr>
<td>1</td>
<td>F_21</td>
<td>Jan-10-2001</td>
</tr>
<tr>
<td>2</td>
<td>F_22</td>
<td>Sep-29-1999</td>
</tr>
<tr>
<td>11</td>
<td>F_21</td>
<td>Jun-12-1999</td>
</tr>
<tr>
<td>12</td>
<td>F_22</td>
<td>Jan-7-2000</td>
</tr>
<tr>
<td>14</td>
<td>F_23</td>
<td>Jan-11-2000</td>
</tr>
<tr>
<td>21</td>
<td>F_21</td>
<td>Feb-1-2001</td>
</tr>
<tr>
<td>22</td>
<td>F_21</td>
<td>Mar-03-2001</td>
</tr>
</tbody>
</table>

**Table AE_Record**

<table>
<thead>
<tr>
<th>Medical_rec_no</th>
<th>AE_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_352001</td>
<td>AE_1</td>
</tr>
<tr>
<td>M_362001</td>
<td>AE_2</td>
</tr>
</tbody>
</table>

**Table Medical_rec**

<table>
<thead>
<tr>
<th>Medical_rec_no</th>
<th>Create_date</th>
<th>Sub_type</th>
<th>Folder_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_311999</td>
<td>Jan-1-1999</td>
<td>W</td>
<td>F_21</td>
</tr>
<tr>
<td>M_322000</td>
<td>Feb-2-2000</td>
<td>O</td>
<td>F_21</td>
</tr>
<tr>
<td>M_331998</td>
<td>Nov-10-1998</td>
<td>W</td>
<td>F_22</td>
</tr>
<tr>
<td>M_341999</td>
<td>Dec-20-1999</td>
<td>O</td>
<td>F_22</td>
</tr>
<tr>
<td>M_352001</td>
<td>Jan-15-2001</td>
<td>O</td>
<td>F_21</td>
</tr>
<tr>
<td>M_362001</td>
<td>Feb-01-2001</td>
<td>O</td>
<td>F_21</td>
</tr>
<tr>
<td>M_382001</td>
<td>Feb-22-2001</td>
<td>O</td>
<td>F_23</td>
</tr>
</tbody>
</table>

**Table Ward_rec**

<table>
<thead>
<tr>
<th>Medical_rec_no</th>
<th>Ward_no</th>
<th>Admission_date</th>
<th>Discharge_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_311999</td>
<td>W_41</td>
<td>Jan-1-1999</td>
<td>Mar-20-1999</td>
</tr>
<tr>
<td>M_322000</td>
<td>W_43</td>
<td>Nov-12-1998</td>
<td>Dec-14-1998</td>
</tr>
</tbody>
</table>

**Table Outpatient_rec**

<table>
<thead>
<tr>
<th>Medical_rec_no</th>
<th>OPD_no</th>
<th>Specialty</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_331998</td>
<td>O_51</td>
<td>Heart</td>
</tr>
<tr>
<td>M_341999</td>
<td>O_52</td>
<td>Ophthalmic</td>
</tr>
<tr>
<td>M_382001</td>
<td>O_53</td>
<td>Therapy</td>
</tr>
</tbody>
</table>

**Table Borrow**

<table>
<thead>
<tr>
<th>Borrower_no</th>
<th>Folder_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>F_21</td>
</tr>
<tr>
<td>1</td>
<td>F_22</td>
</tr>
<tr>
<td>2</td>
<td>F_22</td>
</tr>
<tr>
<td>3</td>
<td>F_23</td>
</tr>
<tr>
<td>11</td>
<td>F_21</td>
</tr>
<tr>
<td>12</td>
<td>F_22</td>
</tr>
<tr>
<td>14</td>
<td>F_23</td>
</tr>
<tr>
<td>21</td>
<td>F_21</td>
</tr>
<tr>
<td>22</td>
<td>F_21</td>
</tr>
<tr>
<td>25</td>
<td>F_23</td>
</tr>
</tbody>
</table>

**Step 1. Schema Translation from the Relational Database to the semantic metadata**

After capturing data semantics, we get an overview of the EER model as shown in Figure 5 where C = categorization and G = disjoint generalization.
Figure 5 An EER model for Hospital Database System with delete and create method

Figure 6 Data flow diagram of semantic metadata

The translated OODB schema is shown as follows:

```java
class ApplicationObject
    super: Composite
    description: "Hospital Record tracking application"
    {}
;

class Doctor
    super: ApplicationObject
    description: "Holds details of a doctor"
    {
        class:
        Doctor find(Integer in_no );
        instance:
            Integer Borrower_no unique;
            String Doctor_name;
        }
;

class Other_hospital
    super: ApplicationObject
    description: "Holds details of other hospital borrower"
    {
```
class: Other_hospital find(Integer in_no);
instance:
    Integer Borrower_no unique;
    String Hospital_name;
);

Class Department
super: ApplicationObject
description: "Holds details of a department"
{
    class:
        Department find(Integer in_no);
    instance:
        Integer Borrower_no unique;
        String Department_name;
    }

Class Borrower
super: Doctor, Department, Other_hospital
description: "Holds details of a borrower"
{
    instance:
        String Borrower_name;
        Bag<Borrow> Borrow_rec;
    }

Class Borrow
super: ApplicationObject
description: "Holds details of a borrow transaction"
{
    class:
        Borrow find(Borrower in_b, Record_folder in_p);
    instance:
        Borrower Borrow_rec;
        Record_folder Rec_borrow;
        Bag<Loan_history> Borrow_history;
    }

Class Loan_history
super: ApplicationObject
description: "Holds details of a past borrow records"
{
    class:
        Loan_history find(Borrower in_b, Loan_history in_p);
    instance:
        Date Return_date;
        Borrow Borrow_history;
    }

Class Record_folder
super: ApplicationObject
description: "Holds details of a record folder"
{
    class:
        Record_folder find(Integer in_no);
        Integer Folder_no unique;
        String Location;
        Patient Patient_record;
        Bag<Borrow> Rec_borrow;
        Bag<Medical_rec> Med_folder;
        Bag<Insurance_mn> Coverage;
    }

Class Insurance_cover
super: ApplicationObject
description: "Holds details of a patient insurance record"
{
    class:
        Insurance-cover find(Integer in_no);
    instance:
        Integer Insurance_no unique;
        Patient Coverage;
    }

Class Medical_rec
super: ApplicationObject
description: "Holds details of a Medical record"
{
    class:
        Medical_rec find(Integer in_no);
    instance:
        Integer Medical_no unique;
        Date Create_date;
        String Sub_type;
        Patient Contain_med;
        Record_folder Med_folder;
    }

Class Patient
Description: "Holds details of patient records"
{
    class:
        Patient find(integer in_no);
    Instance:
        String HKID;
        String Patient_name;
        Bag<record_folder> Patient_record;
        Bag<Insurance_cover> Coverage;
        Bag<medical_rec> Contain_Med;
    }

Class Ward_rec
super: Medical_rec;
description: "Holds details of a ward record"
{
    class:
        Ward_rec find(Integer in_no);
    instance:
        Integer Ward_no;
        Date Admission_date;
Step 2. Data capture from the RDB into the ASCII files
ASCII files are formed with one ASCII file for each relation. "***" is a separator within a particular ASCII file to separate the different records. An example of the ASCII file "Doctor" is shown below.

| O:R:Borrower_no:1 |
| R:Doctor_name:Doctor 1 |
| *** |
| O:R:Borrower_no:2 |
| R:Doctor_name:Doctor 2 |

Step 3. Data materialization from the ASCII files to the OODB
After the data capture process, the Target database and created classes as shown in figure 7 where Borrower is a subclass of Doctor, Department and Other_hospital. The same object appears both in its superclass and in its subclass (i.e. Borrower object with the Oid #3:1 is the same object as Doctor object with the Oid #1:1) as shown below:

| Class Doctor (C1) |
|---|---|---|
| Oid | Borrower_no | Doctor_name |
| #1:1 | 1 | Bloor |
| #1:2 | 2 | Smith |
| #1:3 | 3 | Kim |
| #1:4 | 4 | Chitson |
| #1:5 | 5 | Navathe |

| Class Department (C2) |
|---|---|---|
| Oid | Borrower_no | Department_name |
| #2:1 | 11 | X-Ray |
| #2:2 | 12 | Infant |
| #2:3 | 13 | Chest |
| #2:4 | 14 | Skin |
| #2:5 | 15 | Therapy |

| Class Other_hospital(C10) |
|---|---|---|
| Oid | Borrower_no | Hospital_name |
| #10:1 | 21 | MacNeal |
| #10:2 | 22 | Riveredge |
| #10:3 | 23 | StoneTown |
| #10:4 | 24 | NorthCommunity |
| #10:5 | 25 | GoldenPark |

| Class Insurance_cover (C12) |
|---|---|---|
| Oid | Insurance_no | Coverage |
| #12:1 | I_1 | #13:1 |
| #12:2 | I_2 | #13:2 |

| Class AE (C11) |
|---|---|---|
| Oid | Medical_rec_no | AE_no |
| #6:5 | M_352001 | AE_1 |
| #6:6 | M_362001 | AE_2 |

<p>| Class Borrower (C3) |
|---|---|---|
| Oid | Borrower_no | Borrower_name | Borrow_no |
| #1:1 | 1 | Bloor | [#4:1,#4:5] |
| #1:2 | 2 | Smith | [#4:2] |
| #1:3 | 3 | Kim | [#4:8] |
| #2:1 | 11 | X-Ray | [#4:3] |
| #2:2 | 12 | Infant | [#4:4] |</p>
<table>
<thead>
<tr>
<th>Oid</th>
<th>Folder_no</th>
<th>Location</th>
<th>Patient_record</th>
<th>Rec-borrow</th>
<th>Med_folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>#5:1</td>
<td>F_21</td>
<td>Hong Kong</td>
<td>#13:1</td>
<td>[4:1,4:3,4:6,4:7]</td>
<td>[6:1,6:2,6:5,6:6]</td>
</tr>
<tr>
<td>#5:2</td>
<td>F_22</td>
<td>Kowloon</td>
<td>#13:2</td>
<td>[4:2,4:4,4:5]</td>
<td>[6:3,6:4]</td>
</tr>
<tr>
<td>#5:3</td>
<td>F_23</td>
<td>New Territories</td>
<td>#13:3</td>
<td>[4:8,4:9,4:10]</td>
<td>[6:7]</td>
</tr>
</tbody>
</table>

Class Patient (C13)

<table>
<thead>
<tr>
<th>Oid</th>
<th>HKID</th>
<th>Patient_name</th>
<th>Coverage</th>
<th>Contain_med</th>
<th>Patient_record</th>
</tr>
</thead>
<tbody>
<tr>
<td>#13:1</td>
<td>E3766849</td>
<td>Smith</td>
<td>#12:1</td>
<td>[6:1,6:2,6:5,6:6]</td>
<td>#5:1</td>
</tr>
<tr>
<td>#13:2</td>
<td>E8018229</td>
<td>Bloor</td>
<td>#12:2</td>
<td>[6:3,6:4]</td>
<td>#5:2</td>
</tr>
</tbody>
</table>

Class Medical_rec (C6)

<table>
<thead>
<tr>
<th>Oid</th>
<th>Medical_rec_no</th>
<th>Create_date</th>
<th>Sub_type</th>
<th>Med_folder</th>
<th>Contain-med</th>
</tr>
</thead>
<tbody>
<tr>
<td>#6:1</td>
<td>M_311999</td>
<td>Jan-1-1999</td>
<td>W</td>
<td>#5:1</td>
<td>#13:1</td>
</tr>
<tr>
<td>#6:2</td>
<td>M_322000</td>
<td>Feb-2-2000</td>
<td>O</td>
<td>#5:1</td>
<td>#13:1</td>
</tr>
<tr>
<td>#6:3</td>
<td>M_331998</td>
<td>Nov-10-1998</td>
<td>W</td>
<td>#5:2</td>
<td>#13:2</td>
</tr>
<tr>
<td>#6:4</td>
<td>M_341999</td>
<td>Dec-20-1999</td>
<td>O</td>
<td>#5:2</td>
<td>#13:2</td>
</tr>
<tr>
<td>#6:5</td>
<td>M_352001</td>
<td>Jan-15-2001</td>
<td>O</td>
<td>#5:1</td>
<td>#13:1</td>
</tr>
<tr>
<td>#6:6</td>
<td>M_362001</td>
<td>Feb-01-2001</td>
<td>O</td>
<td>#5:1</td>
<td>#13:1</td>
</tr>
<tr>
<td>#6:7</td>
<td>M_382001</td>
<td>Feb-22-2001</td>
<td>O</td>
<td>#5:3</td>
<td>#13:3</td>
</tr>
</tbody>
</table>

Class Ward_rec (C7)

<table>
<thead>
<tr>
<th>Oid</th>
<th>Medical_rec_no</th>
<th>Ward_no</th>
<th>Admission_date</th>
<th>Discharge_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>#6:1</td>
<td>M_311999</td>
<td>W_41</td>
<td>Jan-1-1999</td>
<td>Mar-20-1999</td>
</tr>
<tr>
<td>#6:2</td>
<td>M_322000</td>
<td>W_43</td>
<td>Nov-12-1998</td>
<td>Dec-14-1998</td>
</tr>
</tbody>
</table>

Class Outpatient_rec (C8)

<table>
<thead>
<tr>
<th>Oid</th>
<th>Medical_rec_no</th>
<th>OPD_no</th>
<th>Specialty</th>
</tr>
</thead>
<tbody>
<tr>
<td>#6:3</td>
<td>M_331998</td>
<td>O_51</td>
<td>Heart</td>
</tr>
<tr>
<td>#6:4</td>
<td>M_341999</td>
<td>O_52</td>
<td>Ophthalmic</td>
</tr>
<tr>
<td>#6:7</td>
<td>M_382001</td>
<td>O_53</td>
<td>Therapy</td>
</tr>
</tbody>
</table>

Inverse materializing OODB back to original RDB for validation

A Hospital Database Systems (HDS) is also implemented in the JASMINE ODBMS. The details of the object oriented schema is the same as the materialized OODB schema as derived in the case study from RDB to OODB.

Step 1. Schema translation from OODB to semantic metadata

After doing the seven steps in the semantic discovery, we can recover semantics from OODB and map them into corresponding RDB schema as follows:

The translated relational schema is:

Relation Patient (Patient_Oid, HKID, Patient_name)
Relation Borrower (*Borrower_Oid, Borrower_no, Borrower_name)
Relation Medical_rec (Medical_rec_Oid, Medical_rec_no, Create_date, Sub_type,
Relation Ward_rec (*Patient_Oid, *Record_folder_Oid)
Relation Outpatient_rec (*Medical_rec_Oid, Medical_rec_no, OPD_no, Specialty)
Relation AE_record (*Medical_rec_Oid, AE_no)
Relation Department (Borrower_Oid, Borrower_no, Department_name)
Relation Doctor (Borrower_Oid, Borrower_no, Doctor_name)
Relation Record_folder (Record_folder_Oid, Folder_no, location, *Patient_Oid)
Relation Other-hospital (Borrower_Oid, Borrower_no, Hospital_name)
Relation Loan_history (Loan_history_Oid, Return_date, *Borrow_Oid)
Relation Insurance_cover (Insurance_Oid, Insurance_nm, *Patient_Oid)

Step 2. Data capture from the OODB to the ASCII files
Source data in the Object oriented database are the same as the converted OODB in case study 1. ASCII files are formed, one for each class. An example of the ASCII file “Doctor” is shown below.

<table>
<thead>
<tr>
<th>ASCII File Name: Doctor</th>
</tr>
</thead>
<tbody>
<tr>
<td>O:OID:#1:1</td>
</tr>
<tr>
<td>R:Borrower_no:1</td>
</tr>
<tr>
<td>R:Doctor_name:Doctor 1</td>
</tr>
<tr>
<td>^*^</td>
</tr>
<tr>
<td>O:OID:#1:2</td>
</tr>
<tr>
<td>R:Borrower_no:2</td>
</tr>
<tr>
<td>R:Doctor_name:Doctor 2</td>
</tr>
</tbody>
</table>

Step 3. Data materialization from the ASCII files to the RDB
After data materialization process, the Target database include data as follows:

Table Doctor (D1)

<table>
<thead>
<tr>
<th>Borrower_Oid</th>
<th>Borrower no</th>
<th>Doctor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1:1</td>
<td>1</td>
<td>Bloor</td>
</tr>
<tr>
<td>#1:2</td>
<td>2</td>
<td>Smith</td>
</tr>
<tr>
<td>#1:3</td>
<td>3</td>
<td>Kim</td>
</tr>
<tr>
<td>#1:4</td>
<td>4</td>
<td>Chitson</td>
</tr>
<tr>
<td>#1:5</td>
<td>5</td>
<td>Navathe</td>
</tr>
</tbody>
</table>

Table Other_hospital (D10)

<table>
<thead>
<tr>
<th>Borrower_Oid</th>
<th>Borrower_no</th>
<th>Hospital name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#10:1</td>
<td>21</td>
<td>Mac Neal</td>
</tr>
<tr>
<td>#10:2</td>
<td>22</td>
<td>Riveredge</td>
</tr>
<tr>
<td>#10:3</td>
<td>23</td>
<td>Stone Town</td>
</tr>
<tr>
<td>#10:4</td>
<td>24</td>
<td>North Community</td>
</tr>
<tr>
<td>#10:5</td>
<td>25</td>
<td>Golden Park</td>
</tr>
</tbody>
</table>

Table Doctor (D1)

<table>
<thead>
<tr>
<th>Borrower_Oid</th>
<th>Borrower no</th>
<th>Doctor_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1:1</td>
<td>1</td>
<td>Bloor</td>
</tr>
<tr>
<td>#1:2</td>
<td>2</td>
<td>Smith</td>
</tr>
<tr>
<td>#1:3</td>
<td>3</td>
<td>Kim</td>
</tr>
<tr>
<td>#1:4</td>
<td>4</td>
<td>Chitson</td>
</tr>
<tr>
<td>#1:5</td>
<td>5</td>
<td>Navathe</td>
</tr>
</tbody>
</table>

Table Other_hospital (D10)

<table>
<thead>
<tr>
<th>Borrower_Oid</th>
<th>Borrower_no</th>
<th>Hospital name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#10:1</td>
<td>21</td>
<td>Mac Neal</td>
</tr>
<tr>
<td>#10:2</td>
<td>22</td>
<td>Riveredge</td>
</tr>
<tr>
<td>#10:3</td>
<td>23</td>
<td>Stone Town</td>
</tr>
<tr>
<td>#10:4</td>
<td>24</td>
<td>North Community</td>
</tr>
<tr>
<td>#10:5</td>
<td>25</td>
<td>Golden Park</td>
</tr>
</tbody>
</table>

Table Patient (D13)

<table>
<thead>
<tr>
<th>Patient_Oid</th>
<th>HKID</th>
<th>Patient_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#13:1</td>
<td>E3766849</td>
<td>Smith</td>
</tr>
<tr>
<td>#13:2</td>
<td>E8018229</td>
<td>Bloor</td>
</tr>
<tr>
<td>#13:3</td>
<td>E6077888</td>
<td>Kim</td>
</tr>
</tbody>
</table>

Table Insurance_cover (D12)

<table>
<thead>
<tr>
<th>Oid</th>
<th>Insurance_no</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>#12:1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>#12:2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table AE_Record (D11)

<table>
<thead>
<tr>
<th>Medical_rec_Oid</th>
<th>Medical_rec_no</th>
<th>AE_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>#6:5</td>
<td>M 352001</td>
<td>AE 1</td>
</tr>
<tr>
<td>#6:6</td>
<td>M 362001</td>
<td>AE 2</td>
</tr>
</tbody>
</table>

Table Borrow (D4)

<table>
<thead>
<tr>
<th>Borrow_Oid</th>
<th>Borrower_Oid</th>
<th>Record_folder_Oid</th>
</tr>
</thead>
<tbody>
<tr>
<td>#4:1</td>
<td>#1:1</td>
<td>#5:1</td>
</tr>
<tr>
<td>#4:2</td>
<td>#1:2</td>
<td>#5:2</td>
</tr>
<tr>
<td>#4:3</td>
<td>#2:1</td>
<td>#5:1</td>
</tr>
<tr>
<td>#4:4</td>
<td>#2:2</td>
<td>#5:2</td>
</tr>
<tr>
<td>#4:5</td>
<td>#1:1</td>
<td>#5:2</td>
</tr>
<tr>
<td>#4:6</td>
<td>#10:1</td>
<td>#5:1</td>
</tr>
</tbody>
</table>
The figure 7 shows the data materialization is feasible.

8 Conclusion

This paper offers a methodology translating the schema among different data models using the semantic metadata. The metadata captures the semantics from legacy database. It consists of classes to preserve constraints after the schema translation, and sequential files to store actual data. It invokes the program call to emulate the method of the OODB, and translates legacy database schema into semantic metadata and to another target legacy database schema upon users request. We present a unique semantic metadata to capture data semantics as well as data operation in a metadata which acts as a knowledge-base system by resolving constraints conflicts between different data models in a heterogeneous database environment. After capturing data and their semantics in a metadata, a united data materialization for the RDB and the OODB using the semantic metadata is shown to be feasible.

With captured semantics and data stored in the semantic metadata, the data can be materialized into the relational database or the object oriented database upon users request. The resulting target database maintains the same semantics in the source database. Using the semantic metadata and data contained in the ASCII file, the data can be materialized to the Relational database or the Object oriented database concurrently.

A validation on data materialization by use of information capacity has been done to show that the semantics of a relational database can be recovered and preserved after materialization. Also the original relational database can be recovered from the object-oriented database without any loss of
information. This shows the logical equivalence between the relational database and the materialized object-oriented database. Such validation is needed to convince the user of the correctness of the materialization process and thus suggest that it is feasible to use a single data set for a concurrent relational view and object-oriented view of an object-relational database. The future research of this paper is universal data warehousing.

### References


35. Paolo Atzeni, Riccardo Torlone: MDM: a Multiple-Data-Model Tool for the Management of


