Object-Relational Management of Multiply Represented Geographic Entities

Anders Friis-Christensen
National Survey and Cadastre
Denmark
afc@kms.dk

Christian S. Jensen
Dept. of Computer Science, Aalborg University
Denmark
csj@cs.auc.dk

Abstract

Multiple representation occurs when information about the same geographic entity is represented electronically more than once. This occurs frequently in practice, and it invariably results in the occurrence of inconsistencies among the different representations. We propose to resolve this situation by introducing a multiple representation management system (MRMS), the schema of which includes rules that specify how to identify representations of the same entity, rules that specify consistency requirements, and rules used to restore consistency when necessary. In this paper, we demonstrate by means of a prototype and a real-world case study that it is possible to implement a multiple representation schema language on top of an object-relational database management system. Specifically, it is demonstrated how it is possible to map the constructs of the language used for specifying the multiple representation schema to functionality available in Oracle. Though some limitations exist, Oracle has proven to be a suitable platform for implementing an MRMS.

1. Introduction

Databases that record information about real-world entities with geographical extents, termed geographic databases, are being used quite widely, e.g., in public administration. The same real-world entities are often being recorded in different databases, which support different applications. This phenomenon, termed multiple representation, invariably leads to inconsistencies, which has been recognized as a key problem in geographic database management [3, 11]. Ensuring consistency among databases describing the same entities (termed representation databases) improves the quality and applicability of data. In this paper, we focus on the case where multiple objects representing the same entity have been registered independently. Such objects are typically managed by legacy database systems, which makes it a difficult challenge to ensure consistency among the objects.

A multiple representation management system (MRMS) maintains consistency among multiply represented geographic entities, and it is legacy-friendly by design. This is because no modifications of the underlying legacy database schemas and associated legacy applications are required when introducing multiple representation management.

The MRMS maintains consistency based on a specification, a multiple representation schema (MRSchema), formulated in a so-called multiple representation schema language (MRSL) [6, 7]. The MRSchema specifies matching, consistency, and restoration rules. The MRSL is based on the assumption that objects representing the same entity exhibit semantic similarities that enable us to model their correspondences.

This paper presents a prototype that maintains the consistency among multiply represented entities drawn from a real-world case from the National Survey and Cadastre in Denmark, who manage a range of independent geographic databases describing the same entities. Because the task of maintaining the various representations has become complex and expensive, on-going work seeks to exploit approaches that ensure consistency and update propagation among the different representations. The results presented in this paper are outcomes of this work. The prototype presented in the paper demonstrates that it is possible to maintain consistency among multiply represented entities based on the functionality offered by object-relational DBMSs. It is shown how to design an MRSchema and how to map the constructs of the MRSL to Oracle's object-relational model. It thus demonstrates that it is possible to design and implement an MRMS in a cost-efficient manner.

Multiple geographic representation has been studied for some years. The MurMur project [15] aims at extending commercial data management software (DBMS or GIS) to support multiple representation, which is similar to our goal. However, they focus on aspects other than matching and restoration rules. Other research [11] has focused on the generalization of map features where there is an exact dependency among representation objects, whereas we consider multiple representations from a more general view-

Proceedings of the 15th International Conference on Scientific and Statistical Database Management (SSDBM'03)
1099-3371/03 $17.00 © 2003 IEEE

point. A single multi-scale database that is capable of storing geographic objects with multiple geometries has also been proposed [9]. This approach assumes an integrated database and does not take into account heterogeneous and independent databases. Another line of study has resulted in a multi-scale database that maintains scale-transition relationships between objects at different scales [5]. This work centers on integration and considers only relationships between pairs of objects. In summary, we are not aware of any work that considers modeling multiple representations with matching, consistency, and restoration rules.

Finally, it should be noted that multiple representation databases relate in a more general sense to federated databases [14]. A federated schema usually provides an integrated view of the underlying autonomous databases. In contrast, the main purpose of an MRMS is to maintain consistency. At a more general level, related work can be found in distributed integrity constraint maintenance [8, 10].

The paper is organized as follows. Section 2 describes the overall approach to integrating multiple representations. Section 3 presents the real-world case used in the paper. Section 4 demonstrates how to map the constructs of the MRSL to the object-relational model of Oracle, and Section 5 presents our experiences of the implementation of the MRMS. Finally, Section 6 concludes the paper.

2. Background—Integration Approach

A first approach to integration might be to describe correspondences among objects representing the same entity. Traditional methods of defining correspondences among databases employ data dependency descriptors to describe how objects of a source class in one database are related to objects of a target class in another database. A problem with this method is that it is only possible to specify dependencies between pairs of source and target classes. This may be sufficient in the field of data warehousing and geographic generalization [18], where the target representation classes are controlled by derivation rules. The main problem, however, is when the r-classes constitute a multiply represented entity that depends on more than one class. No common concept exists that binds the classes together and controls the priority and sequence of updates. Another integration approach is to remove inconsistencies in both schema and data. The problem with this approach is that we change the representation databases.

We instead advocate an approach where a new, so-called integration object (i-object) is introduced that represents an entity. The idea is to describe the correspondences among representation objects (r-objects) via their correspondence with the i-object. The correspondence is termed an mr-association. We use the terms i-classes and r-classes for the classes these objects, and their attributes are termed i-attributes and r-attributes. An i-class then represents the concept of a multiply represented entity, and it is associated with more than one r-class. An example is a real-world building (an entity) that is represented in two databases with different resolutions. We will create an i-object for the building entity, and this i-object will have mr-associations to the two r-objects that represent the entity. Using this new approach simplifies the description of correspondences because we only have dependency descriptors between pairs of i-classes and r-classes.

With this approach, there is no reference from an r-object to its i-object or to other r-objects. This is essential when we want to implement an MRMS on top of existing databases systems without needing to change them. This makes the approach legacy-friendly.

As previously described, the correspondence between an i-class and an r-class is given by matching, consistency, and restoration rules. The matching rules specify how to identify the r-objects that represent the same entity, e.g., using a spatial match. The consistency rules describe the exact correspondences between object instances of the i-class and r-classes and are described at two levels: the *object correspondence* (OC) level and the *value correspondence* (VC) level. These mirror the general notions of existence and value dependencies [4, 13]. Finally, the restoration rules specify which actions to be taken if inconsistency occurs. They specify not only the actions needed for restoring consistency, but specify also the conditions that should trigger an update action.

The MRSL has a graphical and a lexical syntax. The former is based on the Unified Modeling Language (UML) [1], and we use the Object Constraint Language (OCL) [17] to specify constraints. A more thorough description of the MRSL is given elsewhere [6, 7]. An MRSL specification, termed an MRSchema, is used to configure the MRMS. The process is depicted in Figure 1, which shows

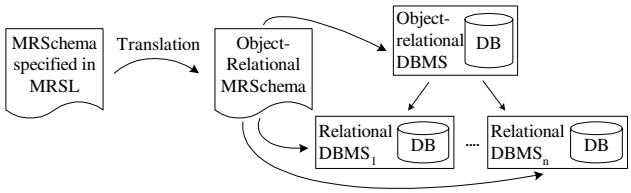

Figure 1. The Implementation Process

that the MRSchema is translated into an object-relational MRSchema, which in turn is implemented in an ORDBMS. Here, the i-objects are stored. We need to extend, but not change, the schemas of the existing databases, because it is necessary to implement triggers, which control updates. Consequently, a requirement to the representation databases is that they support triggers.

3. Case Study

The case study illustrates requirements that need to be supported by an MRMS. The study is based on real-world databases used in the public administration in Denmark that concern buildings. We design an MRSchema that meets the requirements posed in the case study.

3.1. Example Data and Representation Schemas

We consider data from three databases: a topographic map database, a technical map database, and a register database. All three describe the same buildings, but are created and maintained independently by different authorities. The topographic and the technical map databases are created and maintained using relatively costly aerial photo interpretation. One benefit of establishing a correspondence between objects in the two databases is that aerial photo interpretation needs not be duplicated.

Simplified building schemas in UML are seen on Figure 2. (We assume that the reader is familiar with the UML notation [1].) Only representative attributes are given—e.g.,

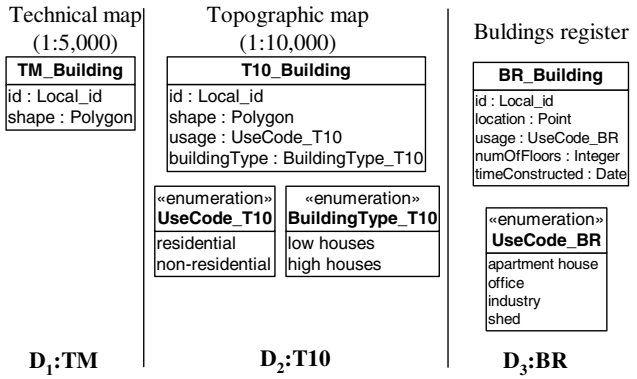

Figure 2. The Three Building Schemas

for a building in the BR database, more than 50 attributes are registered.

The definition of a building varies across the three databases, which complicates the process of creating an MRSchema. The definitions are seen in Table 1. The definitions given here are simplifications of the official ones [2, 12, 16].

An example of the building representations is seen in Figure 3. It is seen that several inconsistencies occur between the shapes of TM and T10 buildings (see the dashed and solid polygons). The shapes of TM buildings are more detailed than those of T10 buildings. Other inconsistencies, not present in the figure, exist in the attributes of the different building objects, e.g., in the usages of the BR buildings and the usage of the corresponding T10 building.

We specify matching, consistency, and restoration requirements for the multiple representation scenario. As the

Table 1. Building Definitions

Class	Definition
TM Building	A building is defined as a building structure on a single property. The shape of a building is stored as a polygon, but very small buildings are usually not registered. All sudden transitions in the building borderline are registered, i.e., the actual and registered borderline coincides.
T10 Building	A building is defined as a detached house, which is larger than 25 square meters. The shape of a building is stored as a polygon, but is registered with as few points as possible. However the difference between the actual and the registered borderline has to be less than 1 meter long.
BR Building	A building in BR is defined as a continuous structure with approximately the same height and style on a single property as the building in TM. A location of a building is stored as a point, but only buildings of a certain size are registered. No exact minimum size exists.

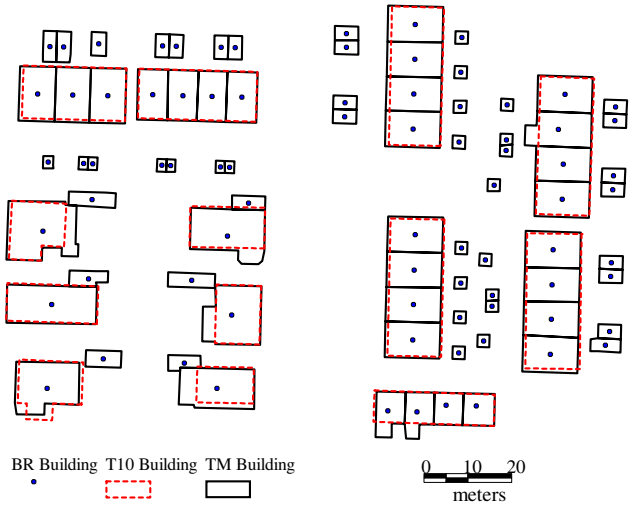

Figure 3. Different Building Representations

three databases were previously independent, we use spatial matching for relating them.

i The location of a BR building is inside the location of the corresponding TM building.

ii To find corresponding TM and T10 buildings, the shapes of the TM and T10 buildings should overlap.

The consistency requirements are the following:

a The shape of T10 building should be a simplification of that of the corresponding TM buildings. That is, the T10 building should be an aggregation that connects overlapping TM buildings. This applies only if the BR building usage is not a shed and the sum of the areas of all corresponding TM buildings is greater than 25m^2.

b The usecode enumeration in a T10 building should correspond to the enumeration in possibly many BR buildings. Since two different value domains exist, transformation from one to the other is needed.

c As the building type of a T10 building is dependent on the height, it should correspond to the number of floors in possibly many BR buildings. Transformation from one value domain to another is also required.

d The location of a BR building should be inside the shape of the corresponding TM building.

e For each TM building there should exist exactly one BR building.

f For each BR building there should exist exactly one TM building.

g For each T10 building there should exist at least one TM and at least one BR building.

When inconsistency is detected, the following restoration actions come into play:

1 If a TM building exists without a corresponding BR building, a BR building needs to be inserted.

2 If a TM and BR building exists (satisfying consistency requirement *a*) without a corresponding T10 building, a T10 building needs to be inserted.

3 If either a BR or T10 building is inserted or exists without a corresponding TM building, it needs to be deleted.

4 If the usage or the number of floors is modified in a BR building, we require the usage or the building type of the T10 building to be modified accordingly.

5 If the location of a BR building is modified it should satisfy the consistency requirement d.

6 If the shape of a TM building is modified, the location of the corresponding BR building and the shape of the corresponding T10 building need to be modified accordingly.

7 If the shape, usage, or building type of a T10 building is modified, consistency requirement a, b, and c, respectively, must be satisfied.

The requirements given above are to be specified in the MRSchema, described next.

3.2. MRSchema

The multiple representation scenario is described in Figure 4. We have created an i-class Building for the TM and BR buildings. As the T10 building is an aggregation of the TM and BR buildings, we create yet another i-class Detached_Building for the T10 building, which is an aggregation of the i-class Building. The matching and restoration rules are specified, using the lexical language, later in this section. The reason for creating two i-classes is that we cannot have an r-class that is an aggregation of i-classes. Because an mr-association is a uni-directional association where only the aggregate knows of its parts, this violates the rule that an r-class should not contain references to its i-class. Creating two i-classes is a more logical approach and, furthermore, the creation of the i-class Detached_Building enables a later extension to, e.g., include a building in the scale of 1:50,000.

The i-class Building is responsible for maintaining the matching requirement i, for consistency requirements d–f, and for restoration actions 1 and 3. The tm role represents the building entity in a technical map database, and the br role represents the building entity in the building register database. The mr-association to the tm building is denoted as master, meaning that it controls the instances of r-objects associated with a given i-object.

The i-class Detached_Building is responsible for maintaining the matching requirement ii, consistency requirements a–c and g, and restoration actions requirements 1–5. The t10 role represents the detached building entity in the topographic map database at scale 1:10,000. The building role represents the building entity based on tm and br building definitions. The mr-association to the building is denoted as master.

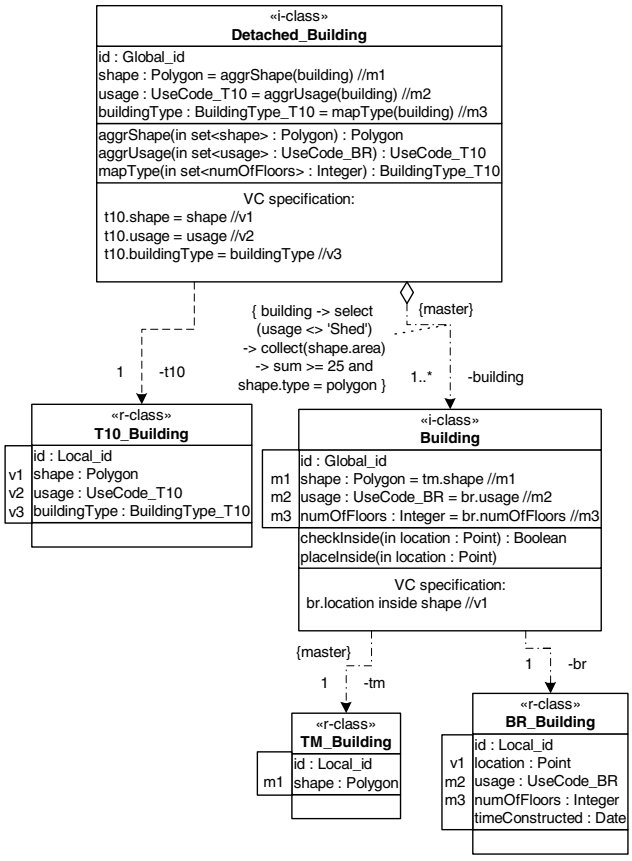

Figure 4. The Building Example in MRSL

Both the i-class Building and the i-class Detached_Building have three attributes that are used for ensuring the consistencies among the r-attributes. These attributes are characteristics of a real-world entity represented by two or more r-classes.

The example shows two common correspondence types for multiple representations of geographic data, the 1:N (or 1:1..N) and 1:1 (or 1:0..1) correspondences. The first will often be of an aggregation type, e.g., a so-called "built-up area" in one database is a composite of several building objects in another. The second is a plain correspondence between two objects, e.g., in a map database and a register database.

The consistency requirements specify that for the i-class `building`, the attribute `shape` has `tm.shape` as master. The master keyword associated with an r-attribute is used to specify that the value of the r-attribute is used when assigning a value to the corresponding i-attribute. The `usage` has `br.usage` as master, and the `numOfFloors` has `br.numOfFloors` as master. The master identifiers (m1–m3) are only included in the design to provide an overview. The operation `checkInside` checks whether `br.location` is inside `tm.shape`.

For the i-class `Detached_Building` the attribute `shape` is an aggregation of one or more `building.shape` attributes (the values of which are initialized from `tm.shape` attribute values), attribute `usage` is an aggregation of `building.usage`, and attribute `buildingType` is an aggregation of `building.numOfFloors`. Again, the master identifiers (m1–m3) are only included for clarity—they do not have any formal meaning. The three operations `aggrShape`, `aggrUsage`, and `mapType` define how the three i-attributes are aggregated from the `Building` i-attributes.

VC `v1` in `Building` specifies that `br.location` values should be inside `building.shape` values. The VCs `v1`–`v3` in `Detached_Building` specify that the r-attributes `t10.shape`, `t10.usage`, and `t10.buildingType` are equal to the corresponding i-attributes of `Detached_Building`.

The i-class `Building` is specified in the lexical language syntax as follows.

```
iclass Building
Attributes:
    id    : Global_id,  shape   : Polygon,
    usage : BR_useCode, numOfFloors : Integer
Operations:
    checkInside(Point) : Boolean,
    placeInside(Point)
Object Correspondence:
    o1: <<master>> tm  [1] : TM_Building,
    o2: br  [1] : BR_Building
Value Correspondence:
    m1: shape = <<master-attribute>> tm.shape,
    m2: usage = <<master-attribute>> br.usage,
    m3: numOfFloors = <<master>> br.numOfFloors,
    v1: shape.checkInside(br.location)
Matching Rules:
    mr1: br.location inside tm.shape,
    mr2: tm.shape cover br.location
Restoration Rules:
    rr1: on insert tm then insert br {immediate},
    rr2: on insert br if not o2 then abort
           {immediate},
    rr3: on update tm.shape if not v1 then
           placeInside(br.location) {immediate},
    rr4: on update br.location if not v1 then
           abort {immediate},
    rr5: on delete tm then delete br {immediate},
    rr6: on delete br if o1 and if not o2 then
           abort {immediate}
end iclass
```

Matching rule `mr1` for the i-class `Building` satisfies matching requirement i. The restoration rules for the i-class `Building` satisfy restoration action requirements 1, 3, 5, and 6. The lexical syntax for the `Detached_Building` is omitted for brevity.

4. Implementation of the MRMS

By mapping the constructs of the MRSL into Oracle9i constructs, this section demonstrates that it is possible to implement the MRMS on top of Oracle. The MRSL is independent of any implementation, and although we present a mapping to Oracle, which is the preferred DBMS at the Danish National Survey and Cadastre, other systems could have been used in the place of Oracle. However, the following implementation details would be different, as different systems provide different object-relational features.

4.1. Choice of Model and Approach

While it is possible to implement the MRSL using only relational constructs, we implement the MRSL using object-relational constructs. This makes it it easier to map the object-oriented MRSL, e.g., object references from the i-objects to their r-objects. When applying the MRMS to already existing databases, it has to be considered whether to convert the existing system to one that uses an object-relational model (assuming it is relational) or to keep it relational and apply the object-relational schema on top. Which approach to choose depends on the existing database and under which circumstance the MRMS is applied. Two situations are common:

- New geographic databases are designed with the aim of storing multiple representations. An example could be the design of a multi-scale system (representations of the same entities at different scales). In this case, there is no preexisting implementation.
- The various representations of the same entities are already implemented in independent relational databases. To ensure consistency among the different representations, the MRMS is implemented on top of existing legacy databases. Our case study is based on this situation, which is common when managing geographic data.

The first situation requires no conversion of existing data, so the object-relational model can be applied directly when designing the databases and implementing the r-classes. In the second situation, two approaches exist: (1) data can be converted from the relational model into the object-relational model so that all r-classes are stored as object tables, or (2) object views can be applied to already existing relational tables.

The advantage of the first approach is that the mapping from the MRSchema is easy and fast. The disadvantage is that the data conversion requires changes to the representation database schemas, which may affect dependent applications. The advantage of the second approach is that the underlying schema is not changed, but data can be accessed as if the tables were defined as object tables. The disadvantage is that subtle differences exist between an object table and an object view, especially in relation to object references. With object tables, references get stored physically in the table. Thus, an attribute in a column defined as a reference type contains a pointer to an object.

However, Oracle does not allow storage of a reference pointing to a virtual object, i.e., an object in a view, because the reference to a virtual object is of a different type than a reference to an object from an object table. This means that it is not possible to integrate object views and object tables when references to virtual objects are required in the object tables. This is problematic for us because we require storage of references from an i-object to its r-objects; and if object views are applied to relational tables, this is not possible. A work-around is to create an object view of the relevant object table using a special Oracle built-in function, `MAKE_REF`, which creates a reference to an object, but this approach complicates the schema unnecessarily.

We show the differences between object tables and object views applied to relational tables in Sections 4.2 and 4.3.

4.2. Preparation of Representation Classes

Before mapping the MRSchema to Oracle, the r-classes must be prepared for the object-relational model. As described in the previous section, we can either implement (or keep) the r-classes as relational tables, or we can implement them as object tables.

The preferred solution is to implement the r-classes as object tables or convert existing relational tables to object tables. In both cases, the first step is to create an object type based on the r-class, the attributes of which should correspond to the attributes of the r-class. An example is seen next, where an object type `BR` is created using the same attribute types as the `BR_Building` r-class:

```
CREATE TYPE br AS OBJECT(id NUMBER(13), location
MDSYS.SDO_GEOMETRY, usage VARCHAR2(20),
numOfFloors NUMBER(3), timeConstructed DATE);
```

The object type can be used either when creating a new object table for storing the objects or for creating an object view. The creation of an object table uses the following statement: `CREATE TABLE TABLE_NAME OF TYPE_NAME`. In our example, the original data is stored in relational tables, but we can convert the relational tables simply by selecting their records into new object tables. When having a more complex relational schema, this step will be correspondingly complex.

If for some reason we want to keep the records in a relational structure, we apply the object-relational model to the relational using object views. An object type is created as described above and is then used to create an object view. The object view for the `BR_Building` table is created similarly to how a standard view is created:

```
CREATE VIEW br_view OF br WITH OBJECT
IDENTIFIER(id) AS SELECT * FROM br_building;
```

These steps complete the preparation of the r-classes.

4.3. Mapping the I-Class and Mr-Associations

The mapping of an i-class depends on whether the corresponding r-classes are implemented as object tables or as relational tables with object views. If the latter case, the i-class needs also to be implemented as a relational table with an object view based on an object type. (Recall that Oracle does not allow storage of object references to virtual objects.) We show this work-around later in this section. If the r-classes are implemented as object tables, we can also implement the i-class as an object table.

The i-class object type (which is required in both situations) is created using two statements. First, an object type is created specifying attributes and operations using the `CREATE TYPE` statement. Second, the body of the object type is created using `CREATE TYPE BODY`, where the code for the operations is specified. Each mr-association is implemented as an attribute of the i-class, where the attribute is a reference type to the corresponding r-object. This means that references to r-objects are physically stored in the i-class. The translation of the i-class is divided into a total of six steps:

Step 1 An Oracle object is created for each i-class in the MRSchema.

Step 2 All attributes from the i-class are parameters to the `CREATE TYPE` statement. Valid data types are Oracle data types such as `NUMBER` and the built-in geometry type `MDSYS.SDO_GEOMETRY`.

Step 3 In addition to the i-attributes, the mr-associations (the OCs) are mapped as supplementary attributes of the i-classes. Each OC is mapped to a corresponding attribute, which then can be stored as nested tables, so that 1:N associations (or aggregations) can be implemented. The data types of the attributes (OC_1–OC_n) are references to r-objects. These data types can be specified as `CREATE TYPE type_name IS TABLE OF REF`

r-class. This statement creates a type that is a reference to an r-class object table.

Step 4 Further parameters to the CREATE TYPE statement are the operations. These include general operations, e.g., checkConsistency and checkAndRestoreConsistency, and optional, user-defined operations. They are mapped to either member methods, invoked on object instances, or static methods, invoked on the object types. For each OC, an operation that checks if the OC is satisfied is implemented as a member function. For each VC, two methods are implemented: one to check whether inconsistency occurs and one to restore consistency. Again, member functions are used.

An example of mapping the i-class Building and the two mr-associations to BR_Building and TM_Building is shown next. The data types for references to the r-classes are created and then the i-class is specified.

```
CREATE TYPE tm_ref AS TABLE OF REF tm_building;
CREATE TYPE br_ref AS TABLE OF REF br_building;
CREATE TYPE building AS OBJECT(id number(15),
   shape MDSYS.SDO_GEOMETRY, usage VARCHAR2(30),
   numberOfFloors NUMBER(3),
   oc1 TM_REF, oc2 BR_REF,
   MEMBER FUNCTION checkOc1 RETURN BOOLEAN,
   MEMBER FUNCTION checkOc2 RETURN BOOLEAN,
   MEMBER FUNCTION checkVc1 RETURN BOOLEAN,
   MEMBER PROCEDURE checkRestoreVc1,
   MEMBER FUNCTION checkM1 RETURN BOOLEAN,
   MEMBER PROCEDURE checkRestoreM1,
   MEMBER FUNCTION checkM2 RETURN BOOLEAN,
   MEMBER PROCEDURE checkRestoreM2,
   MEMBER FUNCTION checkM3 RETURN BOOLEAN,
   MEMBER PROCEDURE checkRestoreM3,
   MEMBER FUNCTION getTm RETURN tm,
   MEMBER FUNCTION getBr RETURN br);
```

This completes the object type creation. We omit the user-defined operations, checkInside and placeInside, from the MRSchema because Oracle has built-in functions that do the necessary operations. The operations checkM1–checkM3 and checkRestoreM1–checkRestoreM3 are used when the master r-attributes are changed. They simply apply new values to the i-class Building. The member functions getTm and getBr return references to the r-objects stored in the i-object. These functions are explained in Section 4.4.

The restore operations can be implemented in either the i-class or in the triggers controlling the restoration actions. We have decided to place the check and restore operations for the VCs in the i-class, but not the operations for restoring the OCs. Though it seem more logical to implement the OCs in the i-class, there are several disadvantages. In particular, when a new r-object, and thus a new i-object, is inserted, we cannot call a method to check the OC in the same transaction because the i-object is first inserted after the trigger has committed. We can create another trigger that finds the i-object again and then calls the method to restore the OC, but this complicates the restoration actions unnecessarily. Thus, we implement the operations in the triggers controlling the restoration, to be described in Section 4.5.

After creating the object type, the next step is to create the body of the object.

Step 5 All methods in the i-class are specified using the CREATE TYPE BODY i-class statement.

An example is shown next for the i-class Building, where we show how the operation checkRestoreVc1 is implemented.

```
CREATE TYPE BODY building AS
  ...
  MEMBER PROCEDURE checkRestoreVc1 IS
  br_id VARCHAR2(13);   rval VARCHAR2(20);
  n_point MDSYS.SDO_GEOMETRY;
  BEGIN
    SELECT br.column_value.id INTO br_id FROM
       TABLE(SELECT oc2 FROM building_tab
       WHERE id = self.id) br;
    IF NOT checkVc1() THEN
        SELECT SDO_GEOM.SDO_POINTONSURFACE
          (b.shape, 0.1) INTO n_point FROM
           building_tab b WHERE b.id=self.id;
        UPDATE br_building br SET br.location=
          n_point WHERE br.id=br_id;
    END IF;
  END;
  ...
END;
```

Here, the built-in function SDO_GEOM.RELATE calculates whether a point is inside a polygon, and the function SDO_GEOM.SDO_POINTONSURFACE creates a new point within a polygon.

The next step is to create the actual tables where the i-objects are stored. The approach varies depending on whether the r-classes are implemented as relational tables or as object tables.

Step 6a If the r-classes are implemented as object tables, a table for storing the i-objects is created. The two reference columns (oc1 and oc2) are stored as nested tables, using NESTED TABLE attribute STORE AS attribute_table, which specifies the tables in which the references to the r-objects are stored.

Step 6b If the r-classes are implemented in relational tables, object views based on each r-class object type are created. Then a relational table is created for each i-class, and in addition a table for each r-class containing i-class id and r-class id is created for storing the references from the i-class to its r-classes. Finally, an object view for the i-class table is created based on the i-class object type and the records in the i-class relational table.

An example of creating an object view based on the i-class type Building with records from the i-class relational table Building_tab and with references to r-objects is seen next:

```
CREATE VIEW building_view OF building WITH
OBJECT OID(id) AS SELECT b.id, b.shape, b.usage,
b.numOfFloors, CAST(MULTISET(SELECT MAKE_REF
(tm_view, tm.tm_id) FROM building_oc1 tm WHERE
b.id=tm.id) AS tm_ref), CAST(MULTISET(SELECT
MAKE_REF(br_view, br.br_id) FROM building_oc2 br
WHERE b.id=br.id)AS br_ref) FROM building_tab b;
```

The statement creates an object view from which we can select object references as in an object table. However, when new i-objects are created, we need to insert them into the relational table.

This completes the creation of the i-class and mr-associations, including the tables to store the i-objects.

4.4. Mapping Matching Rules

The matching of objects that represent the same entity are needed in two situations: when previously independent databases are integrated and when correspondences are already established, but consistency needs to be maintained. The functions needed in the first situation are based on the matching rules specified in the MRSL. For the second situation, there is only a need to query the i-object for its references to its corresponding r-objects in the other databases. This is required in situations where, e.g., an update of an object in a representation database occurs. Only one step is needed to create the matching functions is the body of the i-class (see also Section 4.3):

Step 1 A function is needed for each r-class, and each should return the associated r-object for a given i-object. Each function should check if a reference to an r-object is already stored with the i-object. If not, the matching rules need to be applied to find a corresponding r-object. This occurs when, e.g., an i-object is created.

An example matching rule states that objects of `tm` and `br` represent the same entity if the `br` object is inside the `tm` object. The matching function is seen next, where a function `getBr` matches and returns a `br` building object, either based on being inside a `tm` building object or by querying the i-object for its reference. The matching criterion uses a built-in Oracle function `SDO_GEOM.RELATE`, which determines whether one object is inside another. If true, the object found is returned.

```
MEMBER FUNCTION getBr RETURN br IS
tm_o tm;   br_o br;
BEGIN
  IF checkOc2 THEN
    SELECT DEREF(value(b)) INTO br_o FROM TABLE
      SELECT oc2 FROM building_tab WHERE
      id=self.id) b;
    RETURN br_o;
  ELSE
    SELECT DEREF(value(t)) INTO tm_o FROM TABLE
      (SELECT oc1 FROM building_tab WHERE
      id=self.id) t;
    SELECT VALUE(b) INTO br_o FROM br_building b
      WHERE SDO_GEOM.RELATE(b.location,
      'INSIDE',tm_o.shape,0.1)='INSIDE';
    RETURN br_o;
  END IF;
END;
```

The above function is simplified from the real function, which handles the cases when either no object or more than one object are found.

4.5. Mapping Restoration Rules

Triggers are executed implicitly when an insert, update, or delete occurs on an associated table or view; thus, they are well suited for implementing restoration rules.

Oracle triggers have some limitations that complicate the mapping from the restoration rules to Oracle. In particular, querying or changing mutating tables (i.e., tables being changed by a triggering statement) are not allowed. In addition, only immediate actions are supported, i.e., triggers are executed immediately after an SQL statement is processed. The first limitation is a problem when the i-classes and r-classes are implemented as object tables, and a reference to an r-object is inserted into an i-object. In Oracle, to be able to create a reference to the r-object, a query on the r-class object table is needed. As the r-class table is mutating, this is not allowed in an ordinary trigger. The reason for not allowing querying or changing mutating tables is that the trigger does not have access to a consistent view of the data. The problem with the second limitation is that it is not possible to postpone actions and execute them at a predefined, later stage as detached actions.

The mutating table problem may be circumvented by using *row* triggers and *statement* triggers in combination with, e.g., a PL/SQL table. The row trigger is fired once for each row of the table that is affected by the triggering statement. The statement trigger fires once for each statement and not separately for each row affected by the statement. Here, instead of a single `AFTER INSERT` row trigger on an r-class, which fails to query the table and create the object references to be inserted in the i-class, a row trigger and a statement trigger can be used. The row trigger stores the inserted r-objects in, e.g., a PL/SQL table, and a statement trigger updates the i-class table with object references using the r-objects from the PL/SQL table. It is important to notice that the problem of mutating tables occurs only if object references need to be inserted into the i-class object table. If the i-class table is a relational table, a single row trigger is enough to implement a restoration rule.

Another solution is to use the object views on the r-class relational or object tables. This makes it possible to attach a special kind of trigger, called an `INSTEAD OF` trigger, which makes Oracle fire the trigger instead of executing the triggering event. Then it is possible to query the mutating table. The `INSTEAD OF` trigger should only be used in the case where the dependent applications are implemented

to use object views. If not, there is no reason to change existing applications; hence, the row and statement trigger can be used instead.

As a solution to the problem of implementing a detached action, we can create an object table queue to store the objects that need to be inserted into a table at a specified time. Here, we need appropriate methods to ensure that objects in the queue actually do propagate; and actions to be taken if the propagation fails must be devised.

The steps for mapping the restoration rules to Oracle using `INSTEAD OF` triggers are:

Step 1a Create object views on all associated r-classes, if they do not already exist. Any application depending on the representation databases needs to modify these views instead of the underlying r-class tables. (This is possible only when the views are based on one table with the same or fewer attributes. If it is not feasible to change the depending applications, the row and statement triggers have to be used.)

Step 2a For each restoration rule, an `INSTEAD OF` trigger is created with the appropriate event-condition-action statements. The operation to check consistency of the i-class and its associated r-classes is called from the restoration rules.

An example of a restoration rule for an insert on the `TM_view` is seen below.

```
CREATE TRIGGER insert_tm_obj INSTEAD OF INSERT ON
  tm_view FOR EACH ROW
DECLARE
    b_id NUMBER;   br_id VARCHAR2(13);
    n_point MDSYS.SDO_GEOMETRY;
BEGIN
    INSERT INTO tm_obj VALUES
      (:new.id,:new.shape);
    b_id := generate_b_oid();
    br_id := generate_br_id();
    INSERT INTO building_tab VALUES(b_id,
      :new.shape,null,null,tm_ref(),br_ref());
    INSERT INTO TABLE(SELECT b.oc1 FROM
      building_tab b WHERE b.id=b_id) SELECT
      REF(t) FROM tm_obj tm WHERE tm.id=:new.id;
    SELECT SDO_GEOM.SDO_POINTONSURFACE
      (:new.shape, 0.1) INTO n_point FROM tm_obj
      tm WHERE tm.id=:new.id;
    INSERT INTO br_obj VALUES(br(br_id,null,
      null,null,n_point));
END;
```

The restoration rule first executes the triggering event (the insert statement). In order to have the triggering event processed in an `INSTEAD OF` trigger, it has to be stated explicitly in the trigger. Then a new i-object is created and a reference to the new r-object is inserted. Finally, an object of `BR_Building` is inserted.

If ordinary statement and row triggers are used, the steps for mapping the restoration rules to Oracle are:

Step 1b A PL/SQL table is created for storing the r-objects inserted and the i-object ids. It is used when inserting references to the r-objects into the i-object. Instead of a PL/SQL table, a temporary table or a package variable can be used.

Step 2b For each restoration rule, a row and statement trigger is created with the appropriate event-condition-action statements. The row trigger inserts an i-object into the i-class table without references to its r-objects (as this is not possible) and inserts the r-object and the i-object id in the PL/SQL table. The statement trigger selects the r-object from the PL/SQL table and inserts its reference into the i-object.

This implementation requires that an object type for the r-class exists. It is also assumed that the i-class `Building` is implemented as an object table. If this is not the case, only a row trigger needs to be implemented. We omit an example for brevity.

An example illustrating a restoration action is seen in Figure 5, where an object in the `TM` database is updated. MapInfo (version 7) has been used as an application on top of Oracle to manipulate objects. In the example, the shape

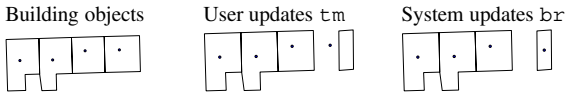

Figure 5. An Update of a Building

of a `tm` building object is made smaller. The update trigger is fired and calls the function `checkRestoreVc1`, which restores the consistency according to the value correspondence. In this case, it means that the location of the `br` building is placed inside the updated `tm` building.

5. Experiences

We proceed to briefly describe our experiences relating to the mapping of the constructs in the MRSL to Oracle.

The overall decisions required during an implementation of an MRMS depend on whether the MRMS is implemented on top of existing legacy database and applications or on top of new databases. In the legacy case, it is typically required that the r-classes are retained as relational tables, to avoid having to change any existing applications. When there are no legacy constraints, we recommend that the r-classes are implemented as object tables. This simplifies the remaining implementation because it can be purely object-relational.

Another decision concerns whether to use ordinary triggers or `INSTEAD OF` triggers to implement restoration rules. The `INSTEAD OF` triggers operate on object views, which means that dependent applications need to update the views instead of the ordinary object/relational tables. This

may be inconvenient when working with legacy systems, in which case ordinary triggers should be used. If the restoration rules are implemented as ordinary triggers, the number of required triggers doubles.

The various problems when implementing the MRMS in Oracle are different in nature. The mutating table problem is inherent and is caused by the implementation of Oracle; still, it is relatively easy to circumvent. The problem of referencing virtual objects can also be circumvented with little effort. A more significant limitation of Oracle is the lack of support for triggers with other than immediate actions. Support for detached actions have to be implemented in the MRMS if needed.

6. Conclusion and Research Directions

Motivated by the problems that occur due to inconsistencies among multiply represented geographical entities, this paper explores the management of multiple representations using object-relational technology. Specifically, the paper argues that it is possible to implement a multiple representation management system (MRMS) based on the object-relational model of Oracle. We have presented an example of the modeling of multiple representations using a real-world case from the Danish public administration. The resulting model, termed a multiple-representation schema, has been mapped to the object-relational model of Oracle, and we have illustrated how it is possible to map the constructs of multiple-representation schemas to database models with the functionality necessary to manage the consistencies among multiple representations.

Oracle has proven itself a suitable platform for implementing a prototype of the MRMS, because of its object-relational model and its support for spatial data types. There are some limitations in Oracle, e.g., in relation to mutating tables and references to virtual objects that unnecessarily complicate the mapping. However, it is possible to circumvent all limitations. Object-relational DBMSs other than Oracle may also be used to avoid some of these problems. We expect no significant difficulties in applying the constructs of the MRSL to other object-relational DBMSs, although the specific will be different.

It is possible to automate some of the steps in the mapping of a multiple-representation schema. Future work includes implementing a parser that semi-automatically translates the MRSchema into, e.g., the object-relational model of Oracle.

Acknowledgments This research was supported in part by the Wireless Information Management network, funded by the Nordic Academy for Advanced Study through grant 000389 and by the LBS project, funded in part by the Danish National Centre for IT Research.

References

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[2] BR. *Circular About the Establishment of the Buildings and Dwellings Register*. Ministry of Housing, January 6, 1977 (in Danish).

[3] B. P. Buttenfield and J. S. DeLotto. Multiple Representations – Scientific Report for the Specialist Meeting. Report 89-3, NCGIA, Department of Geography, SUNY Buffalo, USA, 1989.

[4] S. Ceri and J. Widom. Managing Semantic Heterogeneity with Production Rules and Persistent Queries. In *VLDB*, pp. 108–119, 1993.

[5] T. Devogele, J. Trevisan, and L. Raynal. Building a Multiscale Database with Scale-transition Relationships. In *International Symposium on Spatial Data Handling*, pp. 337–351, 1996.

[6] A. Friis-Christensen, C. S. Jensen, and J. P. Nytun. A Conceptual Schema Language to Manage Multiple Representation of Geographic Entities. Technical report, Aalborg University, Awaiting submission, 2003.

[7] A. Friis-Christensen, D. Skogan, C. S. Jensen, G. Skagestein, and N. Tryfona. Management of Multiply Represented Geographic Entities. In *IDEAS*, pp. 150–159, 2002.

[8] P. W. P. J. Grefen and J. Widom. Protocols for Integrity Constraint Checking in Federated Databases. *Distributed and Parallel Databases*, 5(4):327–355, 1997.

[9] C. Jones, D. Kidner, L. Luo, G. Bundy, and J. Ware. Database Design for a Multi-scale Spatial Information System. *International Journal of Geographic Information Systems*, 10(8):901–920, 1996.

[10] G. Karabatis, M. Rusinkiewicz, and A. Sheth. Interdependent Database Systems, Chapter 8, pp. 217–252. in A. Elmagarmid et al., editors, *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufman, 1999.

[11] T. Kilpeläinen. *Multiple Representation and Generalization of Geo-databases for Topographic Maps*. PhD thesis, Finnish Geodetic Institute, 1997.

[12] KMS. *Top10DK Specification*. National Survey and Cadastre, Copenhagen, 1999. Version 3.1.0 (in Danish).

[13] Q. Li and D. McLeod. Managing Interdependencies among Objects in Federated Databases. In *IFIP Database Semantics Conference on Interoperable Database Systems*, pp. 331–347, 1992.

[14] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[15] S. Spaccapietra, C. Vangenot, C. Parent, and E. Zimanyi. MurMur: A Research Agenda on Multiple Representations. In *International Symposium on Database Applications in Non-Traditional Environments*, pp. 373–384, 1999.

[16] TK99. *Specifications for Techinal Maps TK99*. Committee under the Organization of Municipalities in Denmark, 1999 (in Danish).

[17] J. B. Warmer and A. G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1999.

[18] R. Weibel and G. H. Dutton. Generalising Spatial Data and Dealing with Multiple Representations, *in* P. Longley, M. Goodchild, D. Maguire, and D. Rhind (eds), *Geographic Information Systems - Principles and Technical Issues*, 1999.