Management of Multiply Represented Geographic Entities

Anders Friis-Christensen[1,2] David Skogan [3,4] Christian S. Jensen[1] Gerhard Skagestein [3] Nectaria Tryfona[1,5]

[1]Department of Computer Science, Aalborg University, Denmark, {afc,csj}@cs.auc.dk
[2] Product Development, National Survey and Cadastre, Denmark
[3]Department of Informatics, University of Oslo, Norway, gerhard@ifi.uio.no
[4]SINTEF Telecom and Informatics, Norway, david.skogan@sintef.no
[5]Computer Technology Institute, Athens, Greece, tryfona@cti.gr

Abstract

Multiple representation of geographic information occurs when a real-world entity is represented more than once in the same or different databases. In this paper, we propose a new approach to the modeling of multiply represented entities and the relationships among the entities and their representations. A Multiple Representation Management System is outlined that can manage multiple representations consistently over a number of autonomous databases. Central to our approach is the Multiple Representation Schema Language that is used to configure the system. It provides an intuitive and declarative means of modeling multiple representations and specifying rules that are used to maintain consistency, match objects representing the same entity, and restore consistency if necessary.

Keywords: data modeling, geographic information system, multiple representation, consistency rules, data integration, data management

1 Introduction

Geographic information is needed in a wide range of application domains. This information is often managed independently by various parties and in specialized geographic information systems (GISs). Often, the same real-world *entity* (e.g., a river or a building) is represented by different *objects* in the same or different databases[1]. This phenomenon is called multiple representation, and is a key problem in managing geographic information [3].

Multiple representation may be caused by different approaches in data collection, different semantic definitions, varying levels of detail, or differing application purposes. It may be intended, in that an entity is represented at more than one scale, e.g., the same road can be represented in the scale of 1:10,000 and 1:50,000, or it may be accidental, e.g., when two unrelated databases represent the same entity. In both cases, no tradition exists for maintaining relations among objects representing the same entity, and this may lead to inconsistencies. For example, this happens when a new building is added to a register database, but not to a corresponding topographical map database.

Geographic databases are often developed with a specific application in mind. Data are either captured or copied from another source database and are possibly altered to fit the application. Traditionally, little concern is given to the fact that the source database may change. Since it is not feasible to import the complete source database every time a change occurs, procedures for detecting changes and updating dependent objects should be developed.

Several characteristics of geographic information complicates the process of keeping objects consistent in a multiple representation context. An example is that the spatial extent of an entity may vary depending on the given abstraction. The spatial extent is a complex attribute, e.g., a point, line, or polygon. Consider a city which can be represented as either a point or a polygon depending on the abstraction level used. Fundamental for the representation of geographic information is the scale. Entities may be represented at different scales, which again influences the level of detail of an object being represented. To ensure consistency in a multiple representation context, it is necessary to specify the relationship between the geographic objects. Since these relationships seldom are exact, we need to utilize generalization functions and spatial or topological operators to bridge the gaps among the different abstraction levels. A factor that complicates this process is the varying accuracy of the objects that is to be compared.

The National Survey and Cadastre in Denmark is responsible for a wide range of geographic databases. Previous administrative reorganizations and decisions have led to a range of independent geographic databases describing the same entities at different scales for various juridical and

[1]Here object is used when an entity is represented in a database.

planning purposes. Thus, a need exists for an effective approach to the management of multiply represented geographic entities. To address this need, we introduce the notion of a Multiple Representation Management System (MRMS), the purpose of which is to maintain consistency over selected autonomous databases storing geographic information. The problem of multiple representation is here studied in the context of geographic information. However, it is our belief that the general principles of an MRMS may be applied to other subject areas as well. In connection with the MRMS, we present a Multiple Representation Schema Language (MRSL) that allows users to model multiply represented entities, and to specify consistency and consistency restoration rules in a Multiple Representation Schema (MRSchema). The MRMS operates according to an MRSchema, and it protects prior investments in application development and employee training by operating non-intrusively on top of existing database management systems. A system fulfilling these requirements will ensure geographic representation databases with higher data quality.

Our contributions are threefold:

- We develop a novel concept for modeling multiply represented entities and their consistent representations.
- We outline the Multiple Representation Management System.
- We describe the Multiple Representation Schema Language in detail.

The MRSL is based on an extension to the Unified Modeling Language (UML) and on UML's accompanying Object Constraint Language (OCL).

We assume that the objects representing the same entity (called representation objects) exhibit semantic similarities that enable us to model their correspondences. It is fundamental to our approach to describe how the objects correspond to the entity they represent, rather than to describe the correspondence among objects that represent the same entity. We thus introduce a new type of object that represents the entity (called an integration object), and we describe the correspondence between the integration object and its representation objects. The representation objects can then be seen as "roles" of the integration objects. The need to be able to model multiple representations and roles of geographic entities is described in previous work [9].

Multiple representation of geographic entities has been subject to research especially in the field of cartographic and model based generalization [27]. Kilpeläinen [14] investigates the principles of a system of databases called a multiple representation database. Her work focuses on generalization where there is an exact dependency among representation objects, whereas we focus on multiple representation from a more general point of view.

A subfield of multiple representation databases concerns multi-scale databases. Jones et al. [12] propose a single multi-scale database that is capable of storing geographic objects with multiple geometries. This approach requires an integrated database and does not take into account heterogeneous independent databases. Another approach by Devogele et al. [7] bears similarities with ours. They propose a multi-scale database that maintains scale-transition relationships between objects at different scales. However, they focus mainly on integration and only consider relationships between pairs of objects, whereas our approach handles consistency among more than two representation objects. A current research initiative is the MurMur project, described by Spaccapietra et al. [24, 23]. Its focus is to extend commercial data management software (DBMS or GIS) to support multiple representation, which is similar to our goal. However, no result of this work has been published yet.

The paper is organized as follows. Section 2 presents an example to be used throughout the paper. Section 3 describes the MRMS, and Section 4 describes the MRSL, which is the main contribution of the paper. Finally Section 5 concludes and identifies future work.

2 Case Study

The following section presents an example that illustrates the challenge of managing multiple representation. Excerpts of three databases with their respective schemas in UML are shown in Figure 1. We assume that the reader is familiar with the UML notation [2].

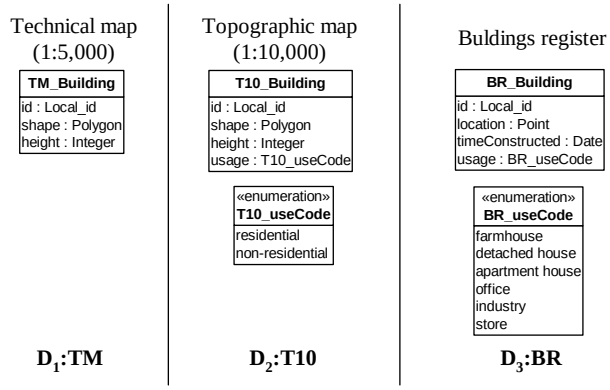

Figure 1. Representation Classes

The scopes of the different databases are described as the follows:

- The technical map database (TM) is used as a digital basis for administrative considerations and register information. It is based on aerial photo interpretation and

on-site registrations. All buildings greater than $10m^2$ are stored as polygons. Updates happen regularly and whenever administrative changes occur.

- The topographic map database, 1:10,000, (T10) is used for analysis purposes and furthermore provides source data for the production of maps in smaller scales. It is based on aerial-photo interpretation. All buildings greater than $25m^2$ are stored as polygons, but with less geometric detail than in the technical map database. Buildings smaller than $25m^2$ are not represented. Updates occur every fifth year.
- The buildings register (BR) gives access to various kinds of juridical information about buildings. A building is stored as a point and defined by its usage. Only buildings of a certain size are registered, i.e., small sheds and outbuildings are not registered. Updates occur regularly.

The classes in the three schemas describe the same entity. Because their objects represent an entity in a specific database, we call these classes representation classes (r-classes). The r-classes all have spatial attributes. The `shape` is of the spatial type polygon and the `location` is of the spatial type point. The attribute domains for the `usage` attribute in T10_BUILDING and BR_BUILDING are based on two different enumerations. The `height` attribute in T10_BUILDING is in meters above sea level, whereas the `height` attribute in TM_BUILDING is the exact height of the building. Finally there is an attribute `timeConstructed` in the BR_BUILDING, which tells us when the building was constructed.

The inexpedience in this multiple representation scenario is that the three databases have been developed separately. There is for example no explicit correspondence between the usage of a building in T10 and BR, even though precise information about the usage exists in BR and could be used in T10 instead of using aerial photo interpretation to resolve the usage. If we can describe the exact correspondence and ensure consistency between these two representations, it would save maintenance efforts and would improve data quality.

An example of building objects can be seen in Figure 2. Here we have a building in TM and the same building in T10 with less detail. In BR, two building objects exist because the extent of the building overlaps two different properties. The `usage` in BR is "store" and "office", respectively. In T10 the `usage` is "non-residential". The height of the building location in T10 is 50 meters above sea level, whereas the exact height of the building itself in TM is 5 meters. The example uncovers several potential consistency requirements:

1. The shape of the building in T10 should be a simplification of the corresponding building in TM.

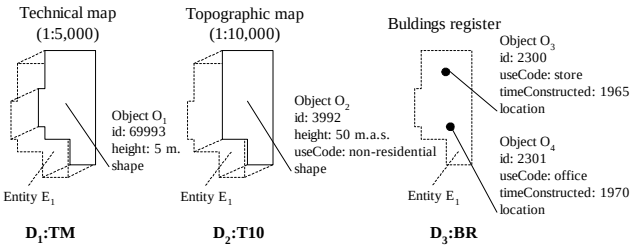

Figure 2. Examples of Building Objects

2. The location of the building in BR should be inside the shape of the corresponding building in TM.
3. The height of the building in T10 should correspond to the sum of the height of the terrain and the building in TM.
4. The usecode enumeration in T10 should correspond to the enumeration in BR. Since two different value domains exist, transformation from one to the other is needed.

These requirements are later specified in a Multiple Representation Schema, which can be used to configure the Multiple Representation Management System described next.

3 Multiple Representation Management System

The aim of this section is to present the motivation and context of our approach. We do this by describing a Multiple Representation Management System (MRMS). It is not our intention to describe the system in detail, but rather to introduce important requirements to such a system.

The main purpose of the MRMS is to actively maintain multiply represented entities with respect to a set of consistency rules. The MRMS can be seen as a loosely coupled, federated database system [21], as it is a collection of cooperating, but autonomous component database systems. The component database systems encapsulate the representation databases. We use the term representation object (r-object) to refer to an object stored in a representation database. The MRMS is loosely coupled since it does not require full control over the representation databases, and neither does it require a complete integrated federated schema. A federated schema usually provides an integrated view of the underlying representation databases. The purpose of the MRMS, however, is different and its schema is called the Multiple Representation Schema (MRSchema). Related work can be found in the field of distributed integrity constraint maintenance [11, 13].

The MRSchema is central to the MRMS. This schema defines a number of consistency, matching, and restoration

rules expressed in the Multiple Representation Schema Language (MRSL) specified in Section 4. The representation databases are assumed to be part of GISs that are maintained by independent parties. As a result, it is technically and politically difficult, if not impossible, to change the schemas or the data manipulation interfaces of the representation databases to accommodate the MRMS. Therefore, the MRMS must operate on top of the representation databases and must be non-intrusive so that the autonomies of the representation databases are maintained.

We identify the following requirements as essential for the management of multiple representations. The MRMS must:

- Be configurable according to the rules defined in the MRSchema.
- Be able to match r-objects located in different representation databases and identify multiply represented entities according to the matching rules stated in the MRSchema.
- Provide a repository over the multiply represented entities in order to keep track of the entities that are consistent and those that are not.
- Evaluate the consistency rules with respect to a multiply represented entity.
- Monitor the representation databases for changes to the r-objects that may affect the consistency of the multiply represented entities managed by the MRMS.
- Send requests for consistency restoration actions to the relevant representation database when an inconsistency is detected. The requests are based on the restoration rules given in the MRSchema.

The MRMS will maintain an acceptable level of consistency among the representation databases. The MRMS also needs efficient access to the rules expressed in the MRSchema to be operational. Since the MRMS is non-intrusive, the consistency restoration actions must be forwarded to the representation databases. Based on the type of the restoration action, it is up to the representation database manager to decide whether the request can be performed automatically or whether it should be handled manually.

Figure 3 shows the architecture of an MRMS. It consists of six components and is associated with a set of component database systems containing representation databases $D_1, \ldots, D_n$. The MRSchema component represents the MRSchema, which defines the multiply represented entities to be managed. The Multiple Representation Engine is the main processing component and is responsible for evaluating the consistency rules given by the MRSchema. It uses the MR Repository to store necessary information about the multiply represented entities managed, and their

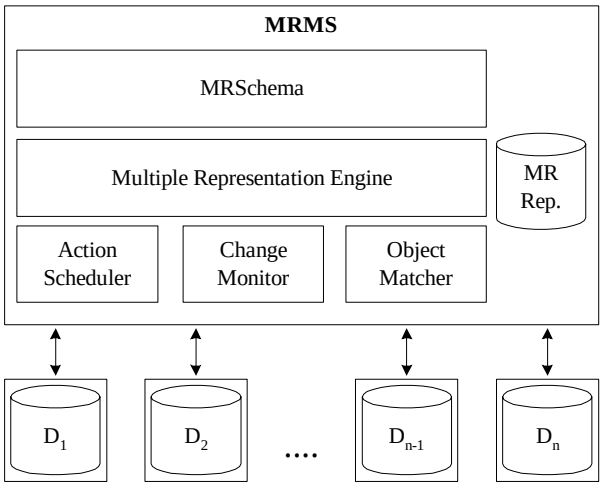

Figure 3. Example MRMS Architecture

current consistency state. The Multiple Representation Engine uses the Object Matcher, Change Monitor and Action Scheduler helper components, to interact with the representation databases. The Object Matcher is responsible for finding corresponding representation objects in the underlying databases that form multiply represented entities. The Change Monitor component is responsible for monitoring the representation databases for changes. If a change occurs, that may lead to an inconsistent state, it notifies the Multiple Representation Engine, which re-evaluates the consistency rules for the respective multiply represented entity. The Action Scheduler is used if an inconsistent is detected. It is mainly responsible for dispatching consistency restoration action requests to the underlying representation databases.

4 Multiple Representation Schema Language

This section presents the multiple representation schema language (MRSL) that is used to model multiply represented entities with consistency rules, matching rules, and restoration rules. We first present the main concepts behind the language. Then the semantics of the most important constructs are explained together with two concrete language syntaxes, a graphical one and a lexical one. The graphical language syntax is based UML and the lexical language syntax is based on OCL [26]. UML and OCL are chosen because of their expressive power and because they can be extended to suit our needs. Further they are platform independent and not tied to a specific implementation. To realize a MRMS system we therefore need to create mappings to specific implementation platforms.

4.1 Towards Integration

To enable a functional MRMS, we need to define pertinent correspondences among the representation databases. Traditional methods of defining correspondences among databases—used, e.g., by Rusinkiewicz et al. [18] and Ceri and Widom [5]—employ data dependency descriptors to describe how objects of a source class are related to objects of a target class in another database. An r-class is typically defined in the context of one database, which stores representation objects (r-objects). It is assumed that r-classes are defined in a common schema language, in our case UML. A dependency from a source class to a target class is denoted by an arrow, which states that a source object is dependent on the existence of a target object and on some of the target object's properties. Figure 4(a) shows a complex dependency scenario. The r-classes R1, R2, and R3 are from different schemas describing a similar concept, i.e., a multiply represented entity. We can see that R2 is dependent on R1 and R3, that R1 is dependent on R2 and that R3 is dependent on R1. If for example R1 or R3 are updated, R2 might need to be updated as well. 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 [27], 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. This is illustrated by the complex dependencies among the three classes shown in Figure 4(a). No common concept exists that binds the classes together and controls the priority and sequence of updates. Thus we cannot maintain multiple representation using dependency descriptors alone.

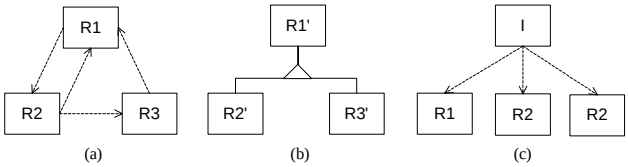

Figure 4. Approaches to Define Correspondence

Another approach is model integration where the aim is to remove inconsistencies in both schema and data. Sheth [20] gives an overview of the interoperability area in which schema [6] and data [7] integration have central parts.

Figure 4(b) illustrates the model integration approach. Our three classes R1, R2, and R3 are here integrated in a class hierarchy by modifying the existing classes resulting in three new classes R1', R2', and R3'. The actual instances then must be converted into the new, integrated schema and inconsistencies resolved.

4.2 Integration Class

The two approaches presented above do not comply with our needs. First, we want to ensure the consistency among multiply represented entities without changing the representation database schemas. Second, we want only binary dependency descriptions to avoid complex correspondence scenarios. The main concept behind our language is the *integration class* (i-class). It allows us to explicitly model multiply represented entities and use them as a basis for defining consistency rules. An i-class is a special class that describes how semantically similar r-classes are related to a multiply represented entity.

Figure 4(c) shows an i-class, I, that controls the dependencies to its r-classes, R1, R2, and R3. It forms a partial integration of its r-classes and constitutes a concept of a multiply represented entity that covers, or includes more than one r-class, because integration is not possible if we have only one r-class. Notice that only the i-class knows of its r-classes—no dependencies among the r-classes exist.

The r-classes associated with the i-class bear similarities to the object-oriented concept of roles [17, 10]. The r-classes can be seen as roles to an i-class and is similar to the description of roles as being adjunct instances, which carries role specific characteristics [25]. In our modeling approach the roles (r-classes) are specified beforehand and then an i-class is created based on a synthesis of the r-classes. This is similar to the role model synthesis in the OOram methodology [16].

An instance of the i-class, the i-object, is a proxy for a multiply represented entity and is responsible for maintaining links to its r-objects and for keeping them consistent with respect to a set of rules. These consistency rules are divided into object correspondences and value correspondences and will be described in Sections 4.3 and 4.4. An i-object is created in a matching process where it is associated with its r-objects. The matching rules will be described in Section 4.5.

Figure 5(a) shows the i-class concept modeled as a metaclass in UML. It defines generic attributes for identifying the i-object, representing object and value correspondences, and for storing matching and restoration rules. It further defines generic operations for evaluating the rules. The metaclass defines a stereotype called `<<i-class>>`, which is used to model user-defined i-classes where application-dependent attributes and operations are specified. An example is shown in Figure 5(b). The i-class' attributes and operations are essential for the specification and evaluation of the consistency rules. Graphically, a new specification compartment for expressing value correspondences is added

to an i-class. This is an extension to the three standard specification compartments of a UML class.

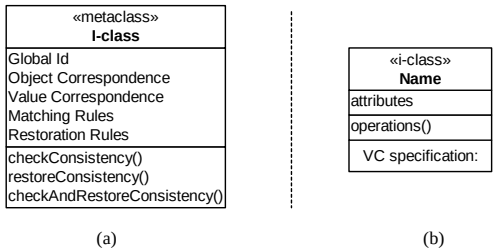

Figure 5. Integration Class

An *multiple representation association* (mr-association) is a directed association from an i-class to an r-class. An i-class has mr-associations to its r-classes to indicate which r-classes are its roles. Similar to the UML association we specify the role name and the multiplicity of the end of an mr-association close to the r-class, as shown in Figure 6. The multiplicity specifies the minimum and maximum number of r-objects needed to form a complete i-object. It is important to emphasize that an mr-association is not an ordinary association, as in the UML terminology, but combines a UML dependency relation (dashed directed line) and an ordinary UML association (normal line). The mr-association is expressed using a dot-dashed, directed line as shown in Figure 6. If the r-class in an mr-association

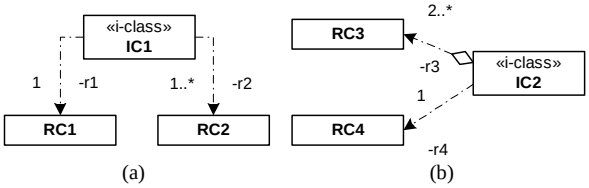

Figure 6. Multiple Representation Associations

is a subordinate concept and perceived as an aggregation, e.g., a built-up area is an aggregation of buildings, this can be symbolized by attaching the UML aggregation symbol to the end of the mr-association, as shown in Figure 6(b).

An *integration attribute* (i-attribute) holds the authoritative value on which value correspondences are evaluated. When an i-class is instantiated, its i-attributes are assigned values from the r-class attributes. It has to be decided which r-attributes that are to be used in this assignment, and the ones chosen are denoted as master r-attributes. The i-attributes are assigned according to the following procedure. Let $\mathcal{A}$ be the set of all attributes of the r-classes that correspond to the i-class. Identify subsets of semantically similar attributes in $\mathcal{A}$. For each subset $A_i \subseteq \mathcal{A}$, define an i-attribute α_i. Define the initial and authoritative value of an i-attribute based on the attributes in A_i. In most cases it can be defined as $\alpha_i = a_j$, where the i-attribute is assigned to exactly one r-attribute. The more general definition is $\alpha_i = f(a_1, \ldots, a_n)$, where $f()$ is a function using more r-attributes $(a_1, \ldots, a_n)$, which all are considered as master attributes to α_i. The type and resolution of the i-attribute are defined by the master r-attribute.

The next two sections describe how to define consistency rules among the i-class and the r-classes at two levels: the object correspondence level and the value correspondence level. These are similar to the existence dependency and value dependency [5, 15].

4.3 Object Correspondence

An *object correspondence* (OC) specifies how an object of one class should be related to an object of another class. Here object correspondences define existence dependencies between the instantiation of the i-class and its associated r-objects. An OC specifies which r-objects that must be present to form a complete i-object, and it is described by the mr-association. The OC can be restricted by attaching constraints to the mr-association indicating that only r-objects satisfying the constraints will form i-objects. OCL is used to specify these constraint, and an example is seen in Figure 7 where a building has two roles to TM_BUILDING and T10_BUILDING. The constraint on the `tm` role specifies that only TM_BUILDINGs with an area larger than or equal to 25 m^2 should be considered. The multiplicity of the mr-associations indicates that we require at least one T10_BUILDING and at least one TM_BUILDING to be present to form an i-object. If the OC is not satisfied

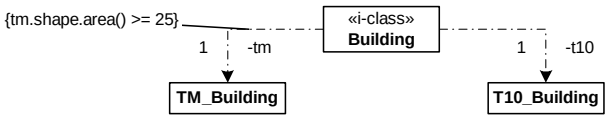

Figure 7. Example of Object Correspondences

then both the i-object and its other corresponding r-objects should be deleted. Alternatively a new r-object could be created to satisfy the OC. See Section 4.6 for a further description of restoration rules. In our lexical language, we can specify the OC as:

```
Object Correspondence:
   tm  [1] : TM_Building
                  { tm.shape.area() >= 25 }
   t10 [1] : T10_Building
```

4.4 Value Correspondence

A *value correspondence* (VC) specifies how attribute values of the i-object and the r-objects should be related. We have previously introduced a special form of value correspondence, which describes how the i-attributes are related to their master r-attributes. Here we describe how the r-attributes, which are not used as master attributes, are related to the i-attributes. In brief, we denote this relationship as $a_j \sim \alpha_i$, where the r-attribute, a_j, has some form of similarity to the corresponding i-attribute, α_i.

A VC is described as an OCL constraint. An OCL constraint is an OCL expression that is Boolean-valued. Thus, if the constraint evaluates to true the VC holds. If the constraint evaluates to false the VC is not satisfied, which indicates that an inconsistency exists. OCL defines a number of different operators. It provides access to object's attributes and operations, as well as navigation of associations and it has mechanisms to iterate over many valued collections. Examples of OCL operators include $\leq, =, \geq$, `forAll`, `exists`, `and`, `or`, and `xor`. We extend these with spatial and topological comparison operators that allows us to compare spatial attributes, e.g., `overlaps`, `inside`, and `touches`. Furthermore it is necessary to provide derivation functions that can be applied to the attributes, e.g., `simplify`(α_i) where `simplify` is a special generalization function that simplifies a curve or a polygon by removing certain points according to a specific algorithm. The most simple form of a VC is of course the comparison operator $=$ with no functions applied to the attributes. A more complex VC can use a combination of the above operations.

Examples of VCs are given in Table 1 that summarizes the VCs of an i-class, I, that integrates the three r-classes R_1, R_2, and R_3. The left hand side of the table defines

Table 1. Value Correspondences

I $(\alpha_1 \ldots \alpha_4)$	R_1 $(a_1 \ldots a_4)$	R_2 $(a_5 \ldots a_8)$	R_3 $(a_9 \ldots a_{12})$
$\alpha_1 = f(a_9, a_{10})$	$a_1 \sim \alpha_1$	$a_5 = \alpha_1$	$\underline{a_9}$
$\alpha_2 = a_6$	$a_2 \sim \alpha_2$	$\underline{a_6}$	$\underline{a_{10}}$
$\alpha_3 = a_{12}$	$\underline{a_3}$	$a_7 \sim \alpha_3$	a_{11}
$\alpha_4 = a_3$	$a_4 = \alpha_4$	$a_8 = \alpha_4$	$\underline{a_{12}}$

four i-attributes $\alpha_1, \ldots, \alpha_4$ and how they are assigned to the respective attributes of the r-classes. The right hand side of the table defines how the r-attributes correspond ($=$ or $\sim$) to the i-attributes. Notice that the VC ($\sim$) needs further specification. In the example, the master of α_1 is defined as a function that aggregates a_9 and a_{10}. The master of α_2, α_3, and α_4 is equal to a_6, a_{12}, and a_3, respectively. The VCs corresponding to R_1's attributes specify that a_1 and a_2 are similar ($\sim$) to α_1 and α_2, respectively, that a_3 is a master (denoted by the underline), and that a_4 has an exact (=) VC to α_4. The rest of the table can be read in the same way. An attribute a_n with no underline is not included as a master attribute, and neither does it have a correspondence to any i-attribute.

If a VC is not satisfied the MRMS needs to restore consistency. If a master r-attribute is changed, the restoration procedure is to change the corresponding i-attribute and then reevaluate the affected VCs. This implies that a comparison operation $=$ will be interpreted as an assignment if the constraint is not satisfied. The restoration procedures will be further described in Section 4.6.

In UML we specify the value correspondence between an i-attribute and its master r-attribute as an initial value in the attribute compartment. A dedicated specification compartment is used to express the other VCs. For the r-class we use a different specification compartment to specify which attributes are used as master and which are included in the VC. Each VC is associated with an identifier. Figure 8 shows an example of an i-class with value correspondences which model the consistency requirements given in Section 2. References to the requirements are given in parentheses. VC `v1` specifies that `t10.shape` must be a simplification of the i-attribute `shape`. The VC is expressed as `t10.shape = gl.simplify(shape)`, where `simplify` is included in a generalization library `gl`. VC `m1` indicates that the `shape` attribute should correspond exactly to the its master value `tm.shape`. VCs `v1` and `m1` jointly satisfy requirement 1. VC `v4` requires that `br.location` is inside `shape` (requirement 2). VCs `m3` and v2 specify that the `height` attribute is assigned to the terrain height of the building plus the buildings height taken from `tm` and furthermore that `t10` height is required to be equal to this value (requirement 3). VCs `m2` and `v3` resolve the possibly many usages of `br` and map this to the `t10` usage (requirement 4).

An example of the VC is given below where the i-class `Building` is specified in the lexical language. In the VC, the `<<master>>` keyword indicates that the i-attribute has the corresponding r-attribute as master.

```
iclass Building
Attributes:
   shape  : Polygon
   usage  : BR_useCode
   height : Integer
Operations:
   resolveUsage(set<BR_useCode>) : BR_useCode
   mapUsage(BR_useCode) : T10_useCode
Object Correspondence:
   tm  [1] : TM_Building
                    { tm.shape.area() >= 25 }
   t10 [1] : T10_Building
   br  [1..*] : BR_Building
Value Correspondence:
   m1: shape = <<master>> tm.shape
   m2: usage = <<master>> resolveUsage(br.usage)
```

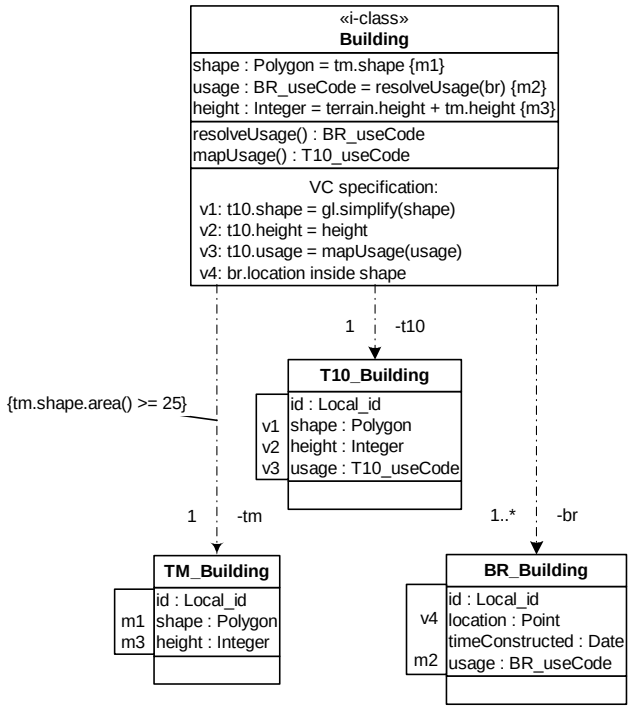

Figure 8. Integration class with VC

```
   m3: height = <<master>> terrain.height(shape)+
                    tm.height
   v1: t10.shape = gl.simplify(shape)
   v2: t10.height = height
   v3: t10.usage = mapUsage(usage)
   v4: br.location inside shape
end iclass
```

4.5 Object Matching Rules

To create an i-object, we need to identify the r-objects that form a multiply represented entity. This process is called object matching, and it is controlled by object matching rules associated with the i-classes. An *object matching rule* specifies a strategy for how to find corresponding r-objects.

The object matching process for an i-class follows certain steps. The first step is to identify the r-objects that are candidates to be matched. The mr-associations together with additional constraints can be used to select the initial set of r-objects, one set for each mr-association. The next step is to select an mr-association as the starting point for the match and to create an incomplete i-object with attributes instantiated with the master r-attributes. Then, for each remaining mr-association, we need to find the r-objects that match the i-object under construction. The process is continued until all r-objects have been associated with an i-object.

Three different matching criteria can be used: global object identifier, attribute comparison, and manual inspection. The *global object identifier* approach provides an exact match criterion. It is assumed that the r-objects share a common unique global identifier that exactly identifies the entity represented. Thus if two or more r-objects have the same global identifier, matching becomes trivial. The need for global identifiers in the management of geographic information is described in previous work [1, 19].

The *attribute comparison* approach involves finding objects based on similar attribute values. The selection of the attributes and the comparison operators can be based on the value correspondences, but also on other criteria. Spatial attributes may provide a basis for matching, because we normally can assume that two objects that are spatially overlapping represent the same entity. The attributes' resolution differences may complicate the matching criteria, and there is a need to define tolerances and operations that take into account the resolution difference between two objects [8, 22]. Attribute comparison may fail due to poor data quality or resolution differences among the r-objects. In such cases the r-objects may still be matched by *manual inspection*. These rules can in their simplest form rely on visual interpretations.

An example is a matching rule for the i-class BUILDING shown in Figure 7. The mr-association to TM_BUILDING is selected as a source, and a set of `tm` buildings greater than 25 m^2 is prepared. For each `tm` building, a match is attempted in the `t10` set using VC `v1` as a matching criterion. After all `tm` objects are associated with an i-object then the possibly remaining unmatched `t10` objects are also associated with an i-object.

The result of the matching process is a set of i-objects, where each i-object is associated with a number of r-objects; the consistency rules can then be evaluated. If the OCs are not satisfied, the i-object is incomplete, i.e., all the required r-objects have not been found. This means that further consistency checks cannot be performed. If the i-object on the other hand has a valid OC then it can be evaluated against its VCs. If the VCs or the OCs are not satisfied we say that the i-object is inconsistent. After the object matching process, we may end up with a set of inconsistent i-objects. In the next section, we describe restoration rules that can be used to restore an i-object's consistency.

4.6 Restoration Rules

If an i-object is inconsistent, we need to identify the reason behind the problem and apply appropriate restoration actions that will restore the i-object to a consistent state. A *restoration rule* specifies not only the actions needed for restoring consistency but also the conditions that should trigger an update action. The restoration rules define the core set of restoration actions that can be applied when an

OC or VC is not satisfied. This may happen when changes occur in the representation databases.

Changes to the r-objects involve insertion of new r-objects and update or deletion of existing r-objects. The restoration actions are highly dependent on the representation databases and what correspondence statements that have been violated. They involve everything from simple requests for actions, such as insert, update, and delete, to complex combinations of such requests. Restoration rules may be automatically generated by analyzing the MRSchema, following the principles described in Ceri and Widom [4].

If an OC is not satisfied, it means that the expected correspondence among the i-object and its r-objects no longer exists. Two strategies can be applied: Either the inconsistent i-object and the remaining r-objects can be deleted, or the absent r-objects can be inserted and then the i-object can be updated. For example if a required r-object has been deleted, a restoration action can be to delete the i-object and its other r-objects altogether. The actions when violating an OC seem straightforward, but the difficult part is to decide which of the two restoration strategies to follow.

If a VC is not satisfied, it means that a value or several of the values of the r-attributes no longer correspond with the i-attributes. There are two ways of restoring an i-object's consistency, either by updating the i-attributes or by changing the i-objects' respective r-objects. An example of the former is if a master r-attribute has changed. Then the respective i-attributes must be updated accordingly. If a VC is violated then the restoration action can be extracted from the VC. This results in a modification request to change the respective r-object.

The restoration actions can either be directly applied to the representation database or they can be applied indirectly, by notifying the representation database manager about the inconsistency and request necessary restoration actions to be made.

An example of a restoration rule for the BUILDING i-class shown in Figure 7 is: if a TM_BUILDING object exists and no corresponding object is found in the T10 representation database, then a T10_BUILDING object shall be created using the VC `v1`.

5 Conclusion

In this paper we have presented a new approach to the modeling and management of multiply represented entities in the context of a Multiple Representation Management System. We have described a Multiple Representation Schema Language (MRSL) that enables users to express their consistency requirements across a system of autonomous databases. Our approach provides a solution to the consistency problems that arise from the multiple representation of entities among geographic information systems.

The MRSL is a comprehensive and expressive language that makes it possible to model multiple representation as well as specifying consistency rules, matching rules, and restoration rules. In addition, it is extendible so that user-defined operations can be specified. A key concept in our approach is the integration class (i-class). Its use permits us to manage multiply represented entities by integrating heterogeneous representation objects via a single integration object. The i-object is responsible for keeping its r-objects consistent according to the consistency rules given by its class. The i-class concept comprises an intuitive approach to multiple representation modeling and it enables us to model the consistency requirements described in the case study. Compared to the traditional dependency description approaches, the i-class reduces the complexity of handling correspondences among several semantically similar r-classes.

Our approach is non-intrusive, which fulfills the requirement that the databases are to be kept autonomous. We use UML and OCL as the basis for our graphical and lexical language syntaxes, and we have found that the extension mechanisms in UML have been sufficient for our needs.

Future work involves formalization of the MRSL, where matching rules and restoration rules must be addressed in particular. We further want to investigate methods to ensure correctness of the rules specified in the MRMS to avoid discrepancies among them. Finally, to test the efficacy of our approach, we need to design and implement a prototype MRMS and evaluate it with respect to a more comprehensive case study.

Acknowledgements

This work was supported in part by a mobility scholarship from the Nordic Academy for Advanced Study (NorFA), the Norwegian Research Council's DYNAMAP-I project 118048/223, the Wireless Information Management network, funded by NorFA through grant 000389, and by a grant from the Nykredit corporation. The authors wish to thank Bjørn Skjellaug for helpful comments.

References

[1] Y. Bishr. A Global Unique Persistent Object ID for Geospatial Information Sharing. In *Proceedings of the 2nd International Conference on Interoperating Geographic Information Systems*, Volume 1580 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1999.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Object Technology Series. Addison-Wesley, USA, 1st edition, 1999.

[3] B. P. Buttenfield and J. S. DeLotto. Multiple Representations – Scientific Report for the Specialist Meeting. Report 89-3, National Center for Geographic Information Analysis, NCGIA, Department of Geography, SUNY at Buffalo, Buffalo, NY 14260, 18-21 Februrary 1989.

[4] S. Ceri and J. Widom. Deriving Production Rules for Constraint Management. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 566–577, Brisbane, Queensland, Australia, 13-16 August 1990. Morgan Kaufmann.

[5] S. Ceri and J. Widom. Managing Semantic Heterogeneity with Production Rules and Persistent Queries. In *Proceedings of the 19th International Conference on Very Large Data Bases*, pages 108–119, Dublin, Ireland, 24–27 Aug. 1993.

[6] T. Devogele, C. Parent, and S. Spaccapietra. On Spatial Database Integration. *International Journal of Geographic Information Systems*, 12(4):335–352, 1998.

[7] T. Devogele, J. Trevisan, and L. Raynal. Building a Multiscale Database with Scale-transition Relationships. In *Proceedings of the 7th International Symposium on Spatial Data Handling*, pages 337–351, Delft, Netherlands, 1996.

[8] M. J. Egenhofer, E. Clementini, and P. D. Felice. Evaluating Inconsistencies Among Multiple Representations. In *Proceedings of the 6th International Symposium on Spatial Data Handling*, pages 901–920, Edinburgh, Scotland, UK, 1994.

[9] A. Friis-Christensen, N. Tryfona, and C. S. Jensen. Requirements and Research Issues in Geographic Data Modeling. In *Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems*, pages 2–8, Atlanta, Georgia, November 2001.

[10] G. Gottlob, M. Schrefl, and B. Röck. Extending Object-Oriented Systems with Roles. *ACM Transactions on Information Systems*, 14(3):268–296, July 1996.

[11] 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.

[12] 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.

[13] G. Karabatis, M. Rusinkiewicz, and A. Sheth. Interdependent Database Systems. In A. Elmagarmid, M. Rusinkiewicz, and A. Sheth, editors, *Management of Heterogeneous and Autonomous Database Systems*, Data Management Systems, Chapter 8, pages 217–252. Morgan Kaufman, 1999.

[14] T. Kilpeläinen. *Multiple Representation and Generalization of Geo-databases for Topographic Maps*. PhD thesis, Finnish Geodetic Institute, 1997. ISBN 951-711-211-4.

[15] Q. Li and D. McLeod. Managing Interdependencies among Objects in Federated Databases. In *Proceedings of the IFIP Database Semantics Conference on Interoperable Database Systems (DS-5)*, IFIP Transactions A-25, pages 331–347, Lorne, Victoria, Australia, Nov. 1992.

[16] T. Reenskaug. *Working With Objects : The OORAM Software Engineering Method*. Manning Publications Co., USA, 1996.

[17] J. Richardson and P. Schwarz. Aspects: Extending Objects to Support Multiple, Independent Roles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 298–307, Denver, Colorado, USA, May 1991.

[18] M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying Interdatabase Dependencies in a Multidatabase Environment. *Computer*, 24(12):46–53, Dec. 1991.

[19] P. Sargent. Features Identities, Descriptors and Handles. In *Proceedings of the 2nd International Conference on Interoperating Geographic Information Systems*, Volume 1580 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 1999.

[20] A. P. Sheth. Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics. In M. F. Goodchild, M. J. Egenhofer, R. Fegeas, and C. A. Kottman, editors, *Interoperating Geographic Information Systems*. Kluwer, 1998.

[21] 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.

[22] D. Skogan. Managing Resolution in Multi-Resolution Databases. In *Proceedings of the 8th Scandinavian Research Conference*, pages 99–113, Ås, Norway, 2001.

[23] S. Spaccapietra, C. Parent, and C. Vangenot. GIS Databases: From Multiscale to MultiRepresentation. In B. Choueiry and T. Walsh, editors, *Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation*, Volume 1864 of *Lecture Notes in Artificial Intelligence*, pages 57–70. Springer, July 2000.

[24] S. Spaccapietra, C. Vangenot, C. Parent, and E. Zimanyi. MurMur: A Research Agenda on Multiple Representations. In *Proceedings of the International Symposium on Database Applications in Non-Traditional Environments*, Kyoto, Japan, November 1999.

[25] F. Steimann. On the Representation of Roles in Object-Oriented and Conceptual Modelling. *Data & Knowledge Engineering*, 35(1):83–106, 2000.

[26] J. B. Warmer and A. G. Kleppe. *The Object Constraint Language : Precise Modeling with UML*. Object Technology Series. Addison-Wesley, USA, 1st edition, 1999.

[27] R. Weibel and G. H. Dutton. Generalising Spatial Data and Dealing with Multiple Representations. In P. Longley, M. Goodchild, D. Maguire, and D. Rhind, editors, *Geographic Information Systems - Principles and Technical Issues*, Volume 1, pages 125–155. John Wiley & Sons, 2 edition, 1999.