DATA UPDATE ACROSS MULTI-SCALE DATABASES

Hae-kyong Kang¹, Jung-Wook Moon² and Ki-Joune Li²
¹Dept. of Geographic Information Science, ²Dept. of Computer Science,
Pusan National University, 609-735, South Korea. Email: {hkang, lik}@pusan.ac.kr

Abstract

This paper discusses on the update problem of multi-scale databases when the multi-scale databases, which is several spatial databases covering the same geographic area with different scales, are derived from an original one. Although the integrity between original and derived multi-scale databases should be maintained, most of update mechanisms do not respect it since the update mechanisms have assumed that the update of source objects propagates to objects directly derived from the source. In order to maintain the integrity of multi-scale databases during updates, we must propagate updates on a source database to objects derived from both the updated source objects and other related objects. This is an important functional requirement of multi-scale database systems, which has been not supported by existing spatial database systems or geographic information systems. In this paper, we propose a set of rules and algorithms for the update propagation and show a prototype developed on ArcGIS of ESRI. Our update mechanism provides with not only the consistency between multi-scale databases but also incremental updates.

1. INTRODUCTION

Multi-scale databases are a set of spatial databases, covering the same geographic area with different spatial types and scales. Multi-scale databases can be derived automatically from existing spatial databases with constraints. In this case, the consistency between multi-scale databases should be respected and it implies that derivation constraints should be preserved based on (Egenhofer 1997). This paper focuses on the update problem of the derived multi-scale databases to maintain their consistency by preserving the derivation constraints.

Precedent researches (Chen 1989 and 1995) on the update problem of derived data have assumed that the update of source objects propagates to those which are directly derived from the source. For example, suppose that two objects $o_1$ and $o_2$ in the source database are aggregated to $o_1'$ in a derived database as shown by in Figure 1(1). In this case, an update on $o_2$ causes an update only on $o_1'$. On the other hands, updates on a source database propagate to objects derived from both the updated source objects and other related objects on the update problem of multi-scale databases. For instance, the update on $s_3$ in Figure 1(2)-E propagates to not only $s_1'$ but also $s_2'$, which is not derived from $s_3$ in Figure 1(2)-D. In addition, the insertion of new objects $s_3'$ and $s_4'$, and the deletion of an existing object $s_1'$ in Figure 1(2)-F must be performed due to the update of $s_3$. These propagations have to be handled for the consistent multi-scale databases.

However, the update problem of multi-scale databases has been rarely considered in the multi-scale research fields. Thus we consider the update problem of multi-scale spatial objects across multi-scale databases in this paper.
Figure 1: Update comparison of a current traditional database and a multi-scale database

This paper is organized as follows. In section 2, previous researches are briefly investigated. We describe a multi-scale data model, rules and algorithms for the update on multi-scale databases in section 3. A link structure between source objects and derived objects is shown to describe the update rules and algorithms, the correspondence between two databases and derivation processes. We show a prototype based on this link structure in section 4, and conclude this paper in section 5.

2. RELATED WORKS

We focus on the update propagation of multi-scale databases. The update propagation may occur when source databases are updated and the multi-scale databases are derived from the source. Thus our work deals with (a) the derivation of multi-scale databases, (b) the consistency on multi-scale databases, (c) the update of spatial databases, and (d) the update propagation over derived multi-scale databases. We briefly introduce some important researches on this theme (Figure 2).

- **Derivation of Multi-scale Database**: The derivation of a new small-scale database from an existing large-scale database can be realized by cartographic generalization, which is motivated to improve its visual quality by removing conflicts of spatial data. In the context of the cartographic generalization, many researches have proposed a set of operators, such as simplification, and smoothing (McMaster 1992 and Muller 1995). In (Davis 1999) these operators are categorized into map generalization, geometric and spatial analysis operators. When more sophisticated results are requested, a derivation
process of new scale databases needs not only spatial distortion mechanism but also semantic abstraction such as merging objects into one (Richardson 1994, Rigaux 1994, and Peng 1996).

- **Consistency on Multi-scale Database**: Consistency at the database level refers to the lack of any logical contradictions within a data model of reality (Egenhofer 1997). We can eliminate logical contradictions by defining constraints, and a database is considered as inconsistent if objects of the database do not respect the constraints described by data model. From this point of view, the consistency of multi-scale databases implies that (i) derivation constraints are preserved, (ii) derived relations are correct. By (Egenhofer 1993) an evaluation method of inconsistencies among multiple representations is proposed. (Tryfona, 1994) suggested a manner to reason proper relations to multi-scale database. (Egenhofer 1994 and Ubeda 1997) proposed to assess the consistency of topological relations.

- **Spatial Data Update**: (Belussi 2000) proposed an integrity driven system to guarantee the topological integrity of spatial databases. The system prevents from destroying the integrity of spatial databases during updates. In (Peled 1996 and 1998) the update problem of spatial databases is considered in terms of update of topographical maps.

- **Update of Derived Multi-scale Data**: (Kidner 1994) introduced a prototype, which maintains multi-scale databases. The update and retrieval of multi-scale databases are performed by deductive processing, employing rules, reasoned from explicit relations and constraints in source databases. This prototype was improved by (Jones 1996).

3. THE UPDATE OF MULTI-SCALE DATABASE

3.1 A Multi-Scale Data Model

We propose a multi-scale data model (Figure 3) consisting of elements to represent source objects, derived objects, and the derivation information of multi-scale databases. The symbols and their meaning, which will be used in the rest of this paper, are given in Table 1.

![Figure 3: A multi-scale data model.](image)

Details about the elements of the multi-scale data model are as follows. \textbf{MultiScaleFeatureClass} (\(FC_{DRV}\)) is derived from \textbf{SourceFeatureClass} (\(FC_{SRC}\)) by a semantic abstraction. The semantic abstraction in a database represents a group of objects with common properties. The properties include adjacency, proximity, or same attribute values in spatial databases. In terms of the semantic abstraction, we consider only two derivation operators in this paper; \textit{GeoAggregation} merges objects within a proximity distance and \textit{Classification} combines objects with same attribute values.
Table 1. Symbols and Meanings

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC</td>
<td>Feature Class</td>
<td>a set of features; FC SRC: a set of source features; FC DRV: a set of derived features from FC SRC the constraints</td>
</tr>
<tr>
<td>f</td>
<td>A set of features ∈ FC</td>
<td>f: ith feature, f geo: a spatial shape of a feature</td>
</tr>
<tr>
<td>RCDRY</td>
<td>Class Directory Class</td>
<td>a set of relations between FC SRC and FC DRV</td>
</tr>
<tr>
<td>RODRY</td>
<td>Object Directory Class</td>
<td>a set of relations between f SRC and f DRV</td>
</tr>
<tr>
<td>att</td>
<td>Attribute</td>
<td>FC.att: an attribute of FC; f.attj: jth attribute of a feature</td>
</tr>
<tr>
<td>dom(att)</td>
<td>Domain of att</td>
<td>the ranges of attribute values of att</td>
</tr>
<tr>
<td>v</td>
<td>An attribute value</td>
<td>f.a = v: v is an attribute value of f.a</td>
</tr>
</tbody>
</table>

**ClassDirectory** (R CDRY ) represents the relationships between FC SRC and FC DRV. The derivationPredicate represents constraints used in derivation processes. The types of constraints are (i) attribute-specified (FC SRC.att), (ii) attribute-value-specified (FC SRC.att = v), and (iii) value-specified(v) by user. In the next section, we describe a set of rules for each type of constraints because the derivation process of FC DRV is determined due to the constraints in different ways.

### 3.2 Update Propagation Rule

We consider four types of updates: insertion, deletion, geometric change and attribute-value change of objects. The update rules deal with the update propagation of FC SRC into FC DRV and we propose update rules for each update type. In Table 2, Rule1 (R1) handles the update of FC DRV derived from FC SRC with the attribute-specified constrain when f ins is inserted into FC SRC. Each update rule is specified by an algorithm given in appendix 1.

Table 2. Rules and Algorithms for Update Propagation.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Update Type</th>
<th>Rule</th>
<th>Algorithm</th>
<th>Constraint</th>
<th>Update Type</th>
<th>Rule</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute-</td>
<td>Insert</td>
<td>R1</td>
<td>A1</td>
<td>Attribute-</td>
<td>Insert</td>
<td>R5</td>
<td>A1</td>
</tr>
<tr>
<td>Specified</td>
<td>Delete</td>
<td>R2</td>
<td>A2</td>
<td>Value</td>
<td>Delete</td>
<td>R6</td>
<td>A2</td>
</tr>
<tr>
<td>Specified</td>
<td>ChangeGeometry</td>
<td>R3</td>
<td>A3</td>
<td>Specified</td>
<td>ChangeGeometry</td>
<td>R7</td>
<td>A3</td>
</tr>
<tr>
<td>Specified</td>
<td>ChangeAttribute</td>
<td>R4</td>
<td>A1, A2</td>
<td>ChangeAttribute</td>
<td>R8</td>
<td>A1, A2</td>
<td></td>
</tr>
<tr>
<td>Value-</td>
<td>Insert</td>
<td>R9</td>
<td>A4</td>
<td>ChangeGeometry</td>
<td>R11</td>
<td>A4, A5</td>
<td></td>
</tr>
<tr>
<td>Specified</td>
<td>Delete</td>
<td>R10</td>
<td>A5</td>
<td>Specified</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Case 1: Attribute Specified as a derivation constraint**

Let FC DRV be a derived class of FC SRC with a constraint, FC SRC.att .

[**Rule1**] New f ins is inserted into FC SRC. The f ins causes f DRV of which predicate is att = v1 to be merged with f ins , if f ins.att = v1 is in dom(f ins.att ). If f ins.att = v is not in dom(f ins.att ) then f ins is to be inserted into FC DRV.

[**Rule2**] f del is deleted from FC SRC. Then, f DRV of which predicate is att = v is modified.

[**Rule3**] f chgeo is a subset of f src in FC SRC which geometry is changed. The f chgeo causes f DRV from f chgeo to be modified.

[**Rule4**] f chatt is a subset of f src, where f src.att = v original is changed to v change . The f chatt causes f DRV original of which predicate is att = v original to be modified. If f chatt.att = v change is in dom(f src.att ), then f DRV att of which predicate is att = v change is merged with f chatt . If f chatt.att = v change is not in dom(f src.att ) then f chatt is to be inserted into FC DRV.
• Case 2 : Attribute-Value Specified as a derivation constraint
Let \( FC_{DRV} \) be a derived class of \( FC_{SRC} \) by \( FC_{SRC} \cdot att = v \), where \( v \) is a subset of \( \text{dom}(FC_{SRC} \cdot att) \).

[Rule5] New \( f_{\text{ins}} \) of which attribute satisfies \( v \) is inserted into \( FC_{SRC} \). It causes \( f_{\text{ins}} \) to be inserted into \( FC_{DRV} \) (that is, \( f_{\text{drv}} \) which predicate is \( att = v \) to be merged with \( f_{\text{ins}} \)).

[Rule6] \( f_{\text{del}} \) is deleted from \( FC_{SRC} \). Then, \( f_{\text{drv}} \) of which predicate is \( att = v_1 \), is modified.

[Rule7] Change of geometry of \( f_{\text{chgeo}} \) in \( FC_{SRC} \) causes \( f_{\text{drv}} \) derived from \( f_{\text{chgeo}} \) to be modified.

[Rule8] \( f_{\text{chatt}} \) is a subset of \( f_{\text{src}} \) which \( f_{\text{src}} \cdot att = v_{\text{original}} \) is changed to \( v_{\text{change}} \). It causes \( f_{\text{drv} \cdot att} \) of which predicate is \( att = v_{\text{original}} \) to be modified as well as \( f_{\text{drv} \cdot att} \) of which predicate is \( att = v_{\text{change}} \) to be modified.

• Case 3 : A value specified by a user as a derivation constraint
Let \( FC_{DRV} \) be a derived class from \( FC_{SRC} \) by \( v \).

[Rule9] New \( f_{\text{ins}} \) is inserted into \( FC_{SRC} \). If \( f_{\text{ins}} \cup f_{\text{drv}} \) satisfies with \( v \), then new \( f_{\text{drv} \cdot new} \) derived from \( f_{\text{ins}} \cup f_{\text{drv}} \) is inserted into \( FC_{DRV} \). \( f_{\text{drv}} \) is deleted from \( FC_{DRV} \).

[Rule10] Deletion of \( f_{\text{del}} \) from \( FC_{SRC} \) causes \( f_{\text{drv}} \) derived from \( f_{\text{del}} \) to be modified. New \( f_{\text{drv} \cdot new} \) derived from \( f_{\text{src}} \cup f_{\text{del}} \) with \( v \), where \( f_{\text{src}} \) is used in derivation of \( f_{\text{drv}} \), is inserted into \( FC_{DRV} \). Then, \( f_{\text{drv}} \) is deleted from \( FC_{DRV} \).

[Rule11] Change of \( f_{\text{ch}} \) in \( FC_{SRC} \) causes \( f_{\text{drv} \cdot i} \) in \( FC_{DRV} \) derived from \( f_{\text{ch}} \) to be modified as well as \( f_{\text{drv} \cdot j} \) in \( FC_{DRV} \) which is derived from \( f_{\text{ch}} \). That is, \( f_{\text{drv} \cdot new} \) derived from \( f_{\text{src}} \cdot f_{\text{ch}} \) with \( v \) where \( f_{\text{src}} \) is the source \( f_{\text{drv}} \), is inserted into \( FC_{DRV} \). Then \( f_{\text{drv} \cdot i} \) is deleted from \( FC_{DRV} \). New \( f_{\text{drv} \cdot new2} \) derived from \( f_{\text{ch}} \cup f_{\text{drv} \cdot j} \) is inserted into \( FC_{DRV} \), where \( f_{\text{ch}} \cup f_{\text{drv} \cdot j} \) satisfies with the \( v \). \( f_{\text{drv} \cdot i} \) is deleted from \( FC_{DRV} \).

3.3 Simple Example
This example shows that R11 is applied to a multi-scale database. Figure 4 shows a multi-scale data model at instance level.
Given Data Set: A class *BuildingBlock* is derived from a source class *Building* by *GeoAggregation* with a predicate “distance < 50”.

**Update of Source**: fsrc3 is moved into the next of fsrc6.

**Update Propagation**: R11 is employed to update *BuildingBlock* because a constraint type of the derivationPredicate “distance<50”, is a value-specified constraint and the update type is *changeGeometry*. R11 is performed by algorithm 4 and 5. For example, line 9 of algorithm 5 calls GeoAggregation(). As a result, two new objects, \( f_{drv3} \) and \( f_{drv4} \), are inserted into \( FC_{DRV} \) and \( f_{drv1} \) is deleted from \( FC_{DRV} \) in the right figure.

4. A PROTOTYPE

We have been developing a prototype of multi-scale database manager with a spatial library of ESRI (ArcObjects), which is a component of ArcGIS (an ESRI product). Main components of the prototype are as follows (Figure 5).

**Modeling Tool** allows users to customize the derivation process through a graphical interaction. It is possible to specify source classes to be loaded, new classes to be derived, generalization operators and predicates to be used in the process. **Generalization Operators** supports six operators (Kang 2001). The operators generate multi-scale databases and their derivation information. **Directory DB** gives corresponding information between source databases and multi-scale databases. **ClassDirectory** and **ObjectDirectory** classes of the diagram in Figure 3 are stored in the directory DB. **Consistency Checker** assesses the multi-scale databases optionally whether relations are correct or not. **Update Rule** supports update propagation rules, proposed in Section 3. These rules are to be employed by the update propagation manager. **Update Propagation Manager** propagates the update of sources to its derived multi-scale databases by referencing the directory DB and employing update propagation rules.

5. CONCLUSION

In this paper, we considered the update problem of multi-scale databases when the multi-scale databases are derived from an original one. In order to maintain the integrity between original and derived multi-scale databases during updates, we must propagate updates on a source database to objects derived from both the updated source objects and other related objects. Nevertheless, most of update mechanisms do not respect it since the update
mechanisms have assumed that the update of source objects propagates to objects directly. Thus we proposed update rules and algorithms for handling update propagations on multi-scale databases. We have been developing a prototype with ArcObjects of ESRI product. Our update mechanism contributes to the consistency and incremental update of a multi-scale database.

ACKNOWLEDGEMENT

This work was supported by the program for cultivating graduated students in regional strategic industry of Korea Industrial Technology Foundation.

REFERENCES


Appendix 1. Update Propagation Algorithms

**Algorithm 1: Insertion**

Input: \( FC_{SRC}, FC_{DRV}, RC_{DRY}, RO_{DRY} \)

Output: \( FC_{DRV} \), updated

Method:
1. Let \( ListUpdateObj \) be a list of inserted objects (\( f_{ins} \)) into \( FC_{SRC} \)
2. constraint \( C \leftarrow RC_{DRY} \) derivationPredicate
3. While ( \( ListUpdateObj \neq \phi \) ) Do
   - Get \( f_{ins} \) from \( ListUpdateObj \)
   - constraintObj \( \leftarrow RO_{DRY} \) derivationPredicate of \( f_{ins} \)
   - If ( \( v \in \text{dom}(FC_{SRC}, att) \) ) where \( att \) = constraint.C
     - Get \( f_{dev} \) which \( RO_{DRY} \) predicate \( \neq v \)
     - \( FC_{DRV} \leftarrow \text{merge}(f_{dev}, f_{ins}) \)
   - Else If ( \( v \in \text{dom}(FC_{SRC}, att) \) )
     - \( FC_{DRV} \leftarrow \text{insert}(f_{ins}) \)
4. End if
5. End while

**Algorithm 2: Deletion**

Input: \( FC_{SRC}, FC_{DRV}, RC_{DRY}, RO_{DRY} \)

Output: \( FC_{DRV} \), updated

Method:
1. Let \( ListUpdateObj \) be a list of deleted objects (\( f_{del} \)) from \( FC_{SRC} \)
2. Let \( ListSrcObj \) and \( ListObj \) be a list of objects each
3. Let \( f_{dev} \) be a subset of \( FC_{DRV} \) where \( f_{dev} \) is derived from \( f_{del} \)
4. While ( \( ListUpdateObj \neq \phi \) ) Do
   - Get \( f_{del} \) from \( ListUpdateObj \)
   - \( ListSrcObj \leftarrow RO_{DRY} \) srcObjects of \( f_{dev} \)
   - \( ListObj \leftarrow \text{ListUpdateObj} \} \) \( \text{ListSrcObj} \)
   - \( ListUpdateObj \leftarrow \text{ListUpdateObj} \} \) \( \text{ListSrcObj} \)
   - While ( \( ListObj \neq \phi \) ) Do
     - Get \( f_{dev} \) from \( ListUpdateObj \)
     - \( FC_{DRV} \leftarrow \text{union}(\text{ListObj} \cup f_{dev}) \)
4. End while
5. End while

**Algorithm 3: Change Geometry**

Input: \( FC_{SRC}, FC_{DRV}, RC_{DRY}, RO_{DRY} \)

Output: \( FC_{DRV} \), updated

Method:
1. Let \( ListUpdateObj \) be a list of changed objects (\( f_{chggeo} \)) in \( FC_{SRC} \)
2. ListSrcObj be a list of objects.
3. Let \( f_{dev} \) derived from \( f_{chggeo} \) be a subset of \( FC_{DRV} \)
4. While ( \( ListUpdateObj \neq \phi \) ) Do
   - Get \( f_{chggeo} \) from \( ListUpdateObj \)
   - \( ListSrcObj \leftarrow RO_{DRY} \) srcObjects of \( f_{dev} \)
   - \( ListObj \leftarrow \text{ListUpdateObj} \} \) \( \text{ListSrcObj} \)
   - \( FC_{DRV} \leftarrow \text{union}(\text{ListObj} \cup f_{dev}) \)
5. End while
6. End while

**Algorithm 4: Insertion**

Input: \( FC_{SRC}, FC_{DRV}, RC_{DRY}, RO_{DRY} \)

Output: \( FC_{DRV} \), updated

Method:
1. Let \( ListUpdateObj \) be a list of inserted objects (\( f_{ins} \)) in \( FC_{SRC} \) which geometry is changed.
2. Let \( f_{dev} \) be all objects of \( FC_{DRV} \)
3. constraint \( \leftarrow RC_{DRY} \) derivationPredicate
4. While ( \( ListUpdateObj \neq \phi \) ) Do
   - Get \( f_{dev} \) from \( ListUpdateObj \)
   - \( ListSrcObj \leftarrow RO_{DRY} \) srcObjects of \( f_{dev} \)
   - \( ListObj \leftarrow \text{ListUpdateObj} \} \) \( \text{ListSrcObj} \)
   - \( ListUpdateObj \leftarrow \text{ListUpdateObj} \} \) \( \text{ListSrcObj} \)
   - call GeoAggregation(\( ListObj \)
5. End while
6. \( FC_{DRV} \leftarrow \text{union}(f_{ins} \cup f_{dev}) \)

**Algorithm 5: Deletion**

Input: \( FC_{SRC}, FC_{DRV}, RC_{DRY}, RO_{DRY} \)

Output: \( FC_{DRV} \), updated

Method:
1. Let \( ListUpdateObj \) be a list of objects (\( f_{del} \)) deleted from \( FC_{SRC} \)
2. Let \( ListSrcObj \) and \( ListObj \) be a list of objects each.
3. Let \( f_{dev} \) be a subset of \( FC_{DRV} \) where \( f_{dev} \) is derived from \( f_{del} \)
4. While ( \( ListUpdateObj \neq \phi \) ) Do
   - Get \( f_{del} \) from \( ListUpdateObj \)
   - \( ListSrcObj \leftarrow RO_{DRY} \) srcObjects of \( f_{dev} \)
   - \( ListObj \leftarrow \text{ListUpdateObj} \} \) \( \text{ListSrcObj} \)
   - \( ListUpdateObj \leftarrow \text{ListUpdateObj} \} \) \( \text{ListSrcObj} \)
   - call GeoAggregation(\( ListObj \)
5. End while
6. \( FC_{DRV} \leftarrow \text{delete}(f_{dev}) 
7. End while

---

Hae-Kyong Kang, Jung-Wook Moon and Ki-Joune Li