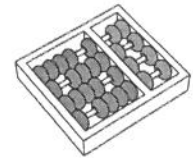João Sávio Ceregatti Longo

"Management of integrity constraints for multi-scale geospatial data"

"*Gerenciamento de restrições de integridade para dados geoespaciais multi-escala*"

CAMPINAS

2013

i

State University of Campinas
Institute of Computing

Universidade Estadual de Campinas
Instituto de Computação

João Sávio Ceregatti Longo

"Management of integrity constraints for multi-scale geospatial data"

Supervisor:
Orientador(a):
Prof.ª Dr.ª Claudia Maria Bauzer Medeiros

"Gerenciamento de restrições de integridade para dados geoespaciais multi-escala"

MSc Dissertation presented to the Post Graduate Program of the Institute of Computing of the State University of Campinas to obtain a Mestre degree in Computer Science.

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

THIS VOLUME CORRESPONDS TO THE FINAL VERSION OF THE DISSERTATION DEFENDED BY JOÃO SÁVIO CEREGATTI LONGO, UNDER THE SUPERVISION OF PROF.ª DR.ª CLAUDIA MARIA BAUZER MEDEIROS.

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA POR JOÃO SÁVIO CEREGATTI LONGO, SOB ORIENTAÇÃO DE PROF.ª DR.ª CLAUDIA MARIA BAUZER MEDEIROS.

_____

Supervisor's signature / Assinatura do Orientador(a)

CAMPINAS

2013

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 13 de Março de 2013, pela

Banca examinadora composta pelos Professores Doutores:
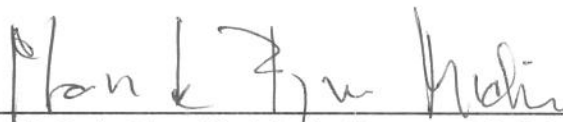
Prof. Dr. Luciano Antonio Digiampietri
EACH / USP

Profª. Drª. Eliane Martins
IC / UNICAMP

Profª. Drª. Claudia Maria Bauzer Medeiros
IC / UNICAMP

Management of integrity constraints for multi-scale geospatial data

João Sávio Ceregatti Longo[1]

March 13, 2013

Examiner Board/*Banca Examinadora*:

- Prof.ª Dr.ª Claudia Maria Bauzer Medeiros (Supervisor/*Orientadora*)

- Prof.ª Dr.ª Eliane Martins
  Institute of Computing - UNICAMP

- Prof. Dr. Luciano Antonio Digiampietri
  Escola de Artes, Ciências e Humanidades - USP

- Prof.ª Dr.ª Ariadne Maria Brito Rizzoni Carvalho
  Institute of Computing - UNICAMP (Substitute/*Suplente*)

- Dr. Alexandre Camargo Coutinho
  CNPTIA - EMBRAPA (Substitute/*Suplente*)

Abstract

Work on multi-scale issues concerning geospatial data presents countless challenges that have been long attacked by GIScience (Geographic Information Science) researchers. Indeed, a given real world problem must often be studied at distinct scales in order to be solved. Another factor to be considered is the possibility of maintaining the history of changes at each scale. Moreover, one of the main goals of multi-scale environments is to guarantee the manipulation of information without any contradiction among the different representations. The concept of scale goes beyond issues of space, since it also applies, for instance, to time. These problems will be analyzed in this thesis, resulting in the following contributions: (a) the proposal of the DBV (Database Version) multi-scale model to handle data at multiple scales from a database perspective; (b) the specification of multi-scale integrity constraints; (c) the implementation of a platform to support model and constraints, tested with real multi-scale data.

Resumo

Trabalhar em questões relativas a dados geoespaciais presentes em múltiplas escalas apresenta inúmeros desafios que têm sido atacado pelos pesquisadores da área de GIS (Sistemas de Informação Geográfica). De fato, um dado problema do mundo real deve frequentemente ser estudado em escalas distintas para ser resolvido. Outro fator a ser considerado é a possibilidade de manter o histórico de mudanças em cada escala. Além disso, uma das principais metas de ambientes multi-escala é garantir a manipulação de informações sem qualquer contradição entre suas diferentes representações. A noção de escala extrapola inclusive a questão espacial, pois se aplica também, por exemplo, à escala temporal. Estes problemas serão analisados nesta dissertação, resultando nas seguintes contribuições: (a) proposta do modelo DBV (Database Version) multi-escala para gerenciar de forma transparente dados de múltiplas escalas sob a perspectiva de bancos de dados; (b) especificação de restrições de integridade multi-escala; (c) implementação de uma plataforma que suporte o modelo e as restrições, testada com dados reais multi-escala.

Acknowledgements

First, I would like to thank my advisor professor Claudia Bauzer Medeiros for the opportunity, help, great advices and corrections.

I would like to thank my mother Isabel, my family, my girlfriend Karina and my friends (in particular Felipe) for all support. Their support was very important for the development of my work.

I would like to thank professor André Santanchè, professor Geneviève Jomier and all members of the Laboratory of Information Systems (LIS) for the many good comments and help to my work.

I would like to thank colleagues, faculty and staff of the Institute of Computing and of UNICAMP, for all the attention and companionship that created a healthy environment for the development of this research.

I would like to thank Dr. Alexandre Continho from CNPTIA – Embrapa, who provided real world multi-scale data and feedback for our case study.

I would also like to thank many other people that were not mentioned but also believed, supported, participated, collaborated with this work and nevertheless remain anonymous in these acknowledgments.

---

[2]Model and Methods in eScience for the Life and Agricultural Sciences

Contents

List of Figures

Chapter 1

Introduction

A major challenge when dealing with geospatial data are the many scales in which such data are represented. For instance, national mapping agencies produce multi-scale geospatial data and one of the main difficulties is to guarantee consistency across the scales [46]. Indeed, geospatial data can be associated with static or dynamic contexts (e.g. mobile applications). In the second case, we can also consider multiple temporal scales, for instance, in units of minutes, hours, days or even years. Temporal scales are important to understand the evolution of a phenomenon or predict what may happen in the future [42].

To clarify the context, let us consider a multi-scale example. Assume a situation in which several groups of people are working in a given geographic region, in a transportation application. In a given spatial scale, say, 1:1000, traffic engineers would analyze local (e.g., street) conditions. In another scale, e.g., 1:100000, the concern would be inter-city transportation. In each scale, new objects (e.g., houses, streets, roads) may be inserted, deleted or modified. This poses several challenges, some of which are attacked here.

It is interesting to materialize multi-scale data in a few choice scales due to modelling requirements and application efficiency, which influence the best scale to be used [2]. Depending on the case, it may be necessary to vary the scales for better data visualization or for different types of analysis, which can result in loss of valuable information. For instance, consider two different spatial scales A and B such that A is larger than B. Each object B will contain one or more corresponding objects of A, but the reverse may be not true [7]. Moreover, inconsistencies may occur by varying the scale. Modifications in objects in a scale (e.g., geometry, localization) can make the data in other scales inconsistent.

Relying on these facts, multi-scale environments should ensure the availability of information without any contradiction across scales [34]. Databases must maintain correct and coherent data, and for this, integrity constraints (ICs) should be defined and checked in update operations to guarantee the quality of data [47, 43]. These issues have been

subject to several research initiatives, that point out many open problems and challenges.

In order to meet some of these challenges, first we propose an approach called DBV (Database Version) multi-scale model [27] to manage multiple scales of geospatial objects. It is based on extending the DBV model [8, 22] to provide support to flexible MRDB (Multi-Representation Databases) structures (data structures to store and link different objects of several representations of the same entity or phenomenon [44]). As will be seen, our extension (and its implementation) provides the following advantages to other approaches: (a) it supports keeping track of evolution of objects at each scale, and across scales, simultaneously; (b) it provides management of multi-scale objects saving storage space [8], as opposed to approaches in which evolution requires replication; and (c) it supports evolution according to scale and to shape, where the latter can be treated as alternative versioning scenarios.

Second, we define some multi-scale integrity constraints (shorthand MS-ICs) to be applied to the DBV multi-scale model. These constraints define conditions under which two states of the world, represented in distinct spatial and/or temporal scales, are jointly consistent.

To validate these ideas, we have implemented a plataform to support the DBV multi-scale model along with the specified MS-ICs. This allows the management of multi-scale data and constraint checking at runtime. Moreover, this plataform has been tested with real multi-scale data.

The goal of this work is investigate the following issues:

- Management of multi-scale data

    - How to manage multi-scale data?
    - How to trace real world evolution?

- Multi-scale consistency

    - How to guarantee multi-scale consistency?

As a consequence, the contributions of this work are the follows:

- the proposal of the DBV multi-scale model;

- the specification of multi-scale integrity constraints;

- the implementation of a platform to support model and constraints, tested with real multi-scale data

This work gave origin to the follow publication: *J. Longo, L. Camargo, C. Medeiros, and A. Santanchè. Using the DBV model to maintain versions of multi-scale geospatial data. In Advances in Conceptual Modeling, volume 7518 of Lecture Notes in Computer Science, pages 284-293. Springer Berlin Heidelberg, 2012.*

Chapter 2

Basic concepts and related work

2.1 Management of multi-scale geospatial data

2.1.1 Overview

Literature on the management of geospatial data at multiple scales concentrates on two directions: (a) generalization algorithms and (b) multi-representation databases (MRDBs). The first are mostly geared towards handling multiple spatial scales via algorithmic processes, that may, for instance, start from predefined scales, or use reactive behaviors (e.g., agents) to dinamically compute geometric properties. MRDBs store data at some predefined scales and link entities of interest across scales, or multiple representations within a scale. These two approaches roughly correspond to Zhou and Jones' [49] multi-representation spatial databases and linked multiversion databases[1].

While generalization approaches compute multiple virtual scales, approaches based on data structures, in which we will concentrate, rely on managing stored data. From this point of view, options may vary from maintaining separate databases (one for each scale) to using MRDBs, or MRMS (Multiple Representation Management Systems) [21]. MRDBs and MRMS concern data structures to store and link different objects of several representations of the same entity or phenomenon [44, 26]. They have been successfully reported in, for instance, urban planning, or in the aggregation of large amounts of geospatial data and in cases that applications require data in different levels of detail [34, 23, 38]. Oosterom et al. [35], in their multi-representation work, also comment on the possibility of storing the most detailed data and computing other scales via generalization. This presents the advantage of preserving consistency across scales (since all except for a basis are computed). Generalization solutions vary widely, but the emphasis is on real time computation, which becomes costly if there are continuous updates to the data – e.g., see

---

[1]We point out that our definition of *version* is not the same as that of Zhou and Jones

4

the hierarchical agent approach of [41] or the multiple representations of [5].

2.1.2   MRDBs

MRDBs (Multiple Representation Databases) are data structures to store and link different objects of several representations of the same entity or phenomenon [44]. There are plenty of benefits to this approach, according to Sarjakoski [44]:

- Maintenance is flexible, since more specific level updates can be propagated to the lower resolution data;

- The links between objects of different levels of representation can provide a basis for consistency and automatic error checking;

- MRDBs can be used for multi-scale analysis of spatial information, such as comparing data at different resolution levels.

According to Deng et al. [14], there are three main variants to link objects in an MRDB. The first one is called *attribute variant* and all data are stored in one dataset. The second variant, named *bottom-up variant*, considers the existence of two or more datasets, linked by an additional attribute that links the objects of the actual scale to those of the immediately smaller scale. The *top-down variant*, the third approach, is similar to the second, except for the fact that the link points to the immediately larger scale.

Moreover, Deng et al. [14] considered linking between different scales in order to establish an MRDB consistent structure. They cited the possible types of links between objects of different scales: 0-to-1, 1-to-0, 1-to-1, n-to-1 and n-to-m. The first value refers to objects of larger scales, while the second refers to the smaller ones. For each type of relationship the authors proposed an approach to solve the inconsistencies. Similarly, this kind of problem was dealt in Bobzien et al.'s work [3].

As an example of implementation, Parent et al. [39] present MurMur, an effort to develop a manipulation approach to geographic databases that have multiple representations. Additional research on MRDB structures includes Burghardt et al.'s work [4], which shows how to improve the creation of maps via automated generalization for topographic maps and multi-representation databases.

Although MRDB structures are used to treat multi-representation problems, this work proposes to deal with multi-scale problems, a subset of those related to multi-representations. Our proposal allows keeping the history of changes within and across scales, which is not directly supported by MRDBs.

2.2 Some issues on scale variation

We have analyzed some consequences of varying spatial and temporal scales. In fact, variations on spatial and temporal scale are best separately represented, "due to the inherently different nature of temporal and spatial dimensions" [7].

2.2.1 Spatial scale variation

Depending on the proximity between geospatial objects, information or relationships between them (*contains*, *overlaps*, *disjoint*, etc) can be easily lost in spatial scale changes [24]. Take the relationship *contains* as an example, and consider the fact that the University of Campinas (UNICAMP) contains many institutes. UNICAMP could be represented in three scales: 1:10k, 1:20k and 1:1M, as Figure 2.1 shows. In the largest scale, UNICAMP represented by a set of polygons while in the smallest there is only a point. This occurs because, as the scale is reduced, the complexity of the objects is reduced and/or some information may be lost. For instance, in the scale 1:20k we do not know what is inside UNICAMP. In other words, the initial relationship "UNICAMP contains institutes" is no longer visible.
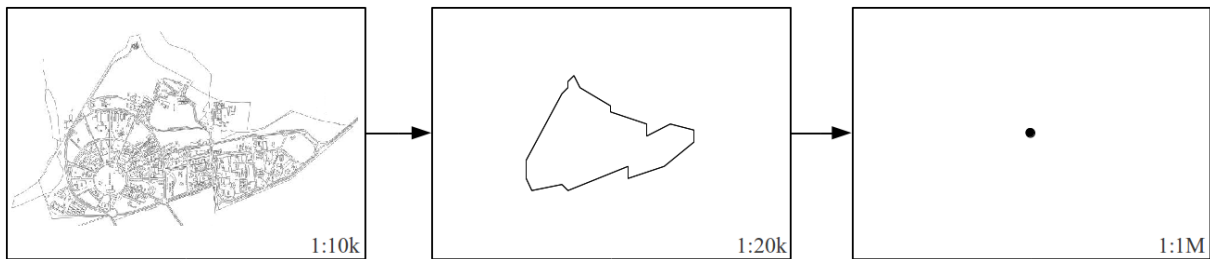
Figure 2.1: UNICAMP represented in three spatial scales

2.2.2 Temporal scale variation

Similarly, important data may be lost by varying the temporal scale. The main example is the summarization of time series, which requires aggregation and elimination of data.

Again, changing scale will eliminate details, outliers, and only aggregate values will be considered.

The same occurs if events along time are distributed separately – e.g., seismic events. Then, their visualization in a map, through time, will either require a video (no temporal aggregation) or the display of aggregate values in a static map.

2.3 The DBV model

The DBV (*Database Version*) model is an approach to "maintain consistency of object versions in multiversion database systems" [8]. A DBV represents a possible state or version of the database [8]. It can be seen as a virtual view of a database, where the database stores multiple versions of objects. This view shows just one version of each object, so that users can work at each DBV as if they were handling a consistent (monoversion) state of the database. Temporal versioning is just one type of version. The DBV model considers a version to be any stored modification of a (database) state. Thus, a given real world object may be versioned in time, but also different simultaneous representations are versions of that object.

In this model, there are two levels: the logical and the physical. The first corresponds to the user view of each database state (DBV) and is represented by the *logical versions*. The second is represented by the *physical versions* of the stored objects.

A *multiversion object* represents one single entity in the real world – any attribute (geometry, color, etc) may change; as long as the experts consider it to be the same entity, it is not assigned a new *id*. Let us consider a *multiversion object o*, e.g. a car, with two different models, one painted blue and other red. Internally, the database will store the *physical versions* of *o* as *pv1* and *pv2*. Logically, *pv1* will appear in one DBV and *pv2* in another. The physical database will have cars of both colors, but from a logical (user's) point of view, only one color exists.

The creation of versions is recorded in a derivation tree as Figure 2.2 shows. Each version is associated with a *stamp* value (0, 0.1, etc). The derivation tree indicates how DBVs are derived from each other, thus supporting change traceability. Derivations always reflect to some kind of update. For instance, Figure 2.2 shows that DBV *d1* (*stamp* 0.1) is derived from *d0* (*stamp* 0) and that *d2* (*stamp* 0.1.1) and *d3* (*stamp* 0.1.2) are derived from *d1*. By definition, there is no data in *stamp* 0 (root).
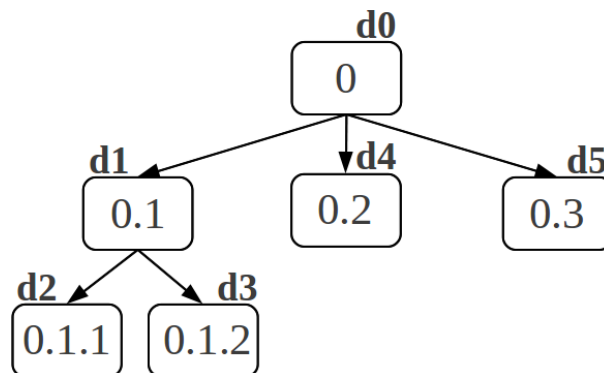
Figure 2.2: Derivation tree of database versions

One of the main advantages of using the DBV approach is that only the changes must be stored. Data that are not modified are shared from previous DBVs through semantics of the version *stamps*. For instance, suppose we have to access all *logical versions* related to *d2*. It is also necessary to look up at all previous DBVs up to the root – *d1*, since each version stores only the data changes. More information about the DBV model can be seen in [8, 22].

2.4 Integrity Constraints (ICs)

Integrity in the context of database systems is a property to ensure that there is no logical contradiction with a model of reality. For this, integrity constraints (IC) should be defined to guarantee that the database understands the semantics required by the model [28].

An integrity constraint is an assertion that intends to prevent the insertion of incorrect data into a database [43].

2.4.1 Specification of integrity constraints

There are several ways to specify integrity constraints – three examples (among many) are: Event-Condition-Action (E-C-A) rules in an active database, ontologies (using some constraint language), or some kind of mathematical formalism.

E-C-A rules are a means of specifying constraints in active databases. Here, the rules are stored with the data, and the database management system monitors Events, upon which Conditions (the constraints) are checked and Actions are taken. Examples of use of E-C-A rules to maintain constraints appear in Medeiros and Cilia [31] and Wang and Reinhardt [48]. The first specified the constraints using logics and the second used *Constraint Decision Table* (CDT). While the E-C-A approach combines constraint declaration with management mechanism, other approaches are more concerned with the specification itself, leaving the maintenance to be treated within an application.

The use of ontologies to specifying constraints appears in, for instance, Mäs et al. [29]. According to the authors, the incorrect use of data may be prevented by using ontologies, which help users in the understanding of concepts and provide a better understanding of the shared data. Thus, they proposed the utilization of SWRL (*Semantic Web Rule Language*), which is a combination of OWL (*Web Ontology Language*) with RuleML (*Rule Markup Language*), to specify ICs.

Mathematical formalism (e.g., logics) are a common means of specifying ICs. An example is Currim and Ram' work [12], which considers that the set of ICs are explicitly modeled during the database conceptual modeling design.

There is also the approach using OCL (Object Constraint Language), which is "a formal language used to describe expressions on UML models" [33]. Some disadvantages of UML are described in [36], such as: "not suitable for automated analysis of verification and validation, etc of architecture" and "UML constructs lack in formal semantics and therefore may become a source of ambiguity, inconsistency in some cases".

Integrity constraints in Multiversion Databases (MVDB) are discussed in [16, 19, 18, 17]. These papers use logics. Basically, an MVDB is consistent if each DBV is consistent and DBVs are mutually consistent [18]. Integrity constraints in an MVDB do not consider ICs across scales, which are the focus of this paper.

In this work, we define constraints using logics, combined with special operators defined by us.

2.4.2 Management of integrity constraints

Management of integrity constraints can be approached at various levels, starting from their specification during database modeling, to maintaining the integrity at the implementation level. This work aims to specify some multi-scale integrity constraints and allow their checking at runtime.

ICs are often translated to database triggers [46, 12, 24], such as in active databases[31] or to metadata stored in a repository [11, 40, 48, 29, 10]. Repositories can be a set of tables, XML files, ontologies, etc. In this work, we chose to implement them directly via methods of classes in Java – see Chapter 4.

2.4.3 Geospatial and temporal integrity constraints

Geospatial constraints are often associated with spatial relationships. According to Guting [25], spatial relationships can be of three kinds – topological, directional and metric. Temporal constraints are related to relationships among time intervals, or an instant and an interval (e.g., an instant is inside an interval, two intervals are disjoint, or one interval comes before/after another interval). Thematic constraints refer to the consistency of thematic attributes (e.g., "buildings should be residential, commercial or industrial"). Finally, there are spatio-temporal constraints, that combine predicates on spatial and temporal attributes [43, 28].

While most papers deal with Spatial/Temporal constraints within a scale, some recent papers have dealt with integrity across (inter) scales. One example is Stoter et al. [46], which proposed a new model to represent multi-scale topography. The authors have discussed ICs across scales, but the focus was on the semantics and they did not consider the temporal scale.

Camossi et al. [6] have studied ICs to preserve object states across temporal scales regarding evolution in time, using E-C-A rules for aggregation and update of data. They do not consider keeping track of data changes (which our model allows), neither do they support alternative representations, as we do.

There are many other kinds of constraints for spatial data in the literature, e.g., the shape constraints of Mäs and Reinhardt [28]. Last but not least, several authors (e.g. Cockroft [10, 9]) propose user constraints, in which users define constraints that are specific to their domain. These constraints are beyond the scope of this work, and are left for future research.

2.4.4   Conclusion

This Chapter presented an overview of basic concepts that will be used in the rest of the text. In particular, it presented two basis for this work – the DBV model, and integrity constraints for multi-scale geospatial data.

Chapter 3

The multi-scale model and inter-scale constraints

3.1 DBV multi-scale model

3.1.1 Overview

We have adopted the DBV model to support multiple scales. Each DBV represents the world in a particular scale. The set of DBVs, which can be interlinked, correspond to a multi-scale/multi-representation world.

We extended the model so that, instead of one derivation tree, each scale has its own tree and all trees evolve together. Besides the version *stamp*, each DBV has now an associated scale s. We use the following notation: dX_s denotes DBV dX of scale s. Figure 3.1 shows the derivation trees for four versions (0, 0.1, 0.1.1 and 0.1.2) and n scales. Notice that all trees have the same topology, thereby indicating that in the real world, when an object is versioned, this happens at all scales.

Let a real world object *o1* be physically stored in a database in two scales, receiving physical identifiers *pv1* and *pv2*, where the geometry of *pv1* is a polygon and *pv2* a point. Polygon and point are respectively represented in DBVs $d1_1$ and $d1_2$. Using this information and the DBV concepts, we have two *logical versions* (each in a DBV) represented in the following way: *logical version 1* = $((o1, d1_1), pv1)$ and *logical version 2* = $((o1, d1_2), pv2)$. In other words, DBV $d1_1$ contains the polygon version of *o1*, and $d1_2$ the point version of *o1*.

Unlike several multi-representation approaches, we do not link explicitly objects of different scales (e.g., *pv1* and *pv2*). Instead, the link is achieved implicitly by combining *stamp* and derivation trees, using the concept of *logical versions*. This kind of link is similar to the *bottom-up variant* seen in section 2.1.2.
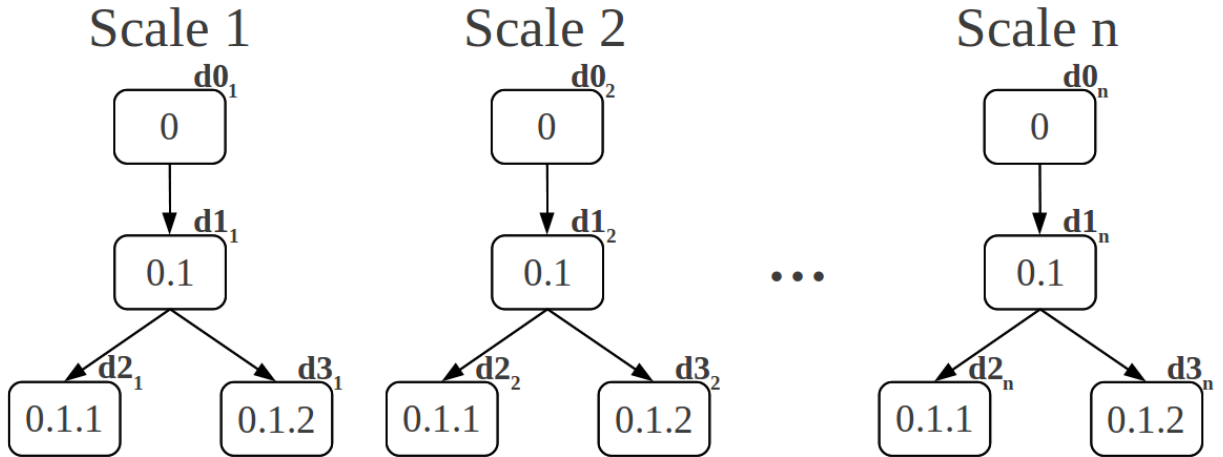
Figure 3.1: Example of our approach to maintain versions of multi-scale geospatial data

A change in the real world that requires creating a new version in scale s may require changes in other scales. Keeping one tree per scale, thus, makes sense because, as remarked by [45], for large scale changes an object suffers radical changes when scale changes occur and thus there is seldom any intersection (if any) between DBVs in different scales. To simplify maintaining consistency across scales, we postulate that all derivation trees grow and shrink together and have the same topology. This leads to the notion of multi-scale *scenario* σ, for short, *scenario*. A *scenario* is formed by all the DBVs with the same version *stamp*. For instance, in Figure 3.1, $d0_1$, $d0_2$, ..., $d0_n$ form a *scenario*, and so do $d1_1$, $d1_2$, ..., $d1_n$; etc. In fact, there may be many *scenarios*.

For managing the versions, we use the propagation algorithm adopted by the DBV model: only data changes must be stored and unchanged data are propagated across versions.

3.1.2 The Model

Figure 3.2 represents our model in UML. We introduce a new class called *Scale*, which has an identifier named *sid* (*scale id*). A DBV is identified by the couple (*stamp*, *sid*). The *Scale* class allows the association of a DBV with different types of scales, where spatial scale is one of them (another example is the temporal scale). A *MultiversionObject* may encapsulate others through aggregations (more details will be seen in section 3.1.4). The *LogicalVersion* class associates a *MultiversionObject* to a *DBV*. A *physical version* of an object underlies a *logical version* (i.e., it may appear in some DBV). This is expressed by the relationship between *LogicalVersion* and *PhysicalVersion* classes. The latter is the root of a hierarchy of classes of all kinds of objects that can be versioned and allows the

user to choose which data will be versioned because *PhysicalVersion* needs to be extended. A DBV has one parent and – by a derivation process – one or more children.

Moreover, if a *multiversion object o* does not appear in DBV *d*, we represent this situation by setting its value (as a *logical version*) as *null*. Thus, initially all objects are stored as *null* in the root DBVs (for convenience, we omitted this fact in the examples).
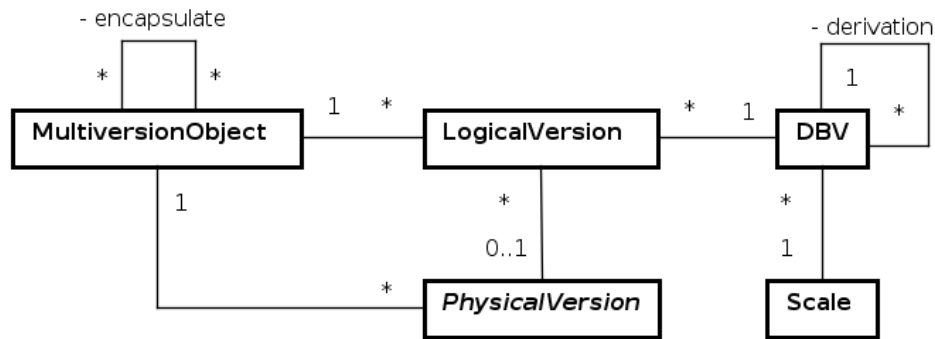
Figure 3.2: Our basic model in UML

The model considers the following operations:

1. **Create/update/remove a *multiversion object***. These operations persist, update and remove a *multiversion object*, respectively;

2. **Create a new scale *s***. This operation persists in the database a *Scale* entity and a root DBV for scale *s*. If other derivation trees already exist, the new tree has to reflect the topology of the other trees by repeatedly invoking operations that create DBVs whose logical states correspond to sets of objects in scale *s*;

3. **Create a DBV *d* from its parent *p***. This operation persists in the database a *DBV* entity *d* derived from *p*. Its initial state is the same as parent *p*;

4. **Add, delete or modify a *logical version* of an object from a DBV**. *Logical versions* are stored as tuples <*dbvid, oid, pvid*>. If an object does not exist in a given DBV, it receives value *null* instead of *pvid*. This operation manipulates *LogicalVersion* entities, adding new or modifying existing ones. Deletion of a *logical version* eliminates the tuple from the table. Users updating a *logical version* are in fact updating a DBV. At each update, users can decide whether this will require versioning. In this case, this is reflected into creating a new *scenario* (operation 7);

5. **Remove a DBV *d***. First, this operation removes the *logical versions* related to *d*. Next, *d* is deleted from the database. This operation can only be accomplished if there are no DBVs derived from *d*;

6. **Access a DBV**. The DBV is made available to the user by collecting all *logical versions* of its objects. I.e., the corresponding view of the multiversion database is made available;

7. **Create a *scenario* by deriving a DBV *da* from *db***. This operation creates a new DBV in all derivation trees, by inserting at each scale s DBV da_s as child of DBV db_s repeatedly invoking the create DBV operation. At the end, the set of created DBVs is considered as the current *scenario*;

8. **Remove a *scenario sc***. This operation removes the DBVs (and all their logical data) which form *sc*. This is only possible if all DBVs of *sc* are leaves regarding their derivation trees;

3.1.3 A simple example

Consider Figure 3.3, where the roots (*stamp* 0) appear for scales 1:10000, 1:20000 and 1:50000. This example concerns urban vectorial data, and the Figure illustrates a given city section. The first version from root (*stamp* 0.1) shows the initial state of the section represented in the three scales. Version 0.1.1 and 0.1.2 show evolution alternatives in that section (either prioritizing the horizontal road, or the vertical road). The geometries in dotted lines represent the propagated data.
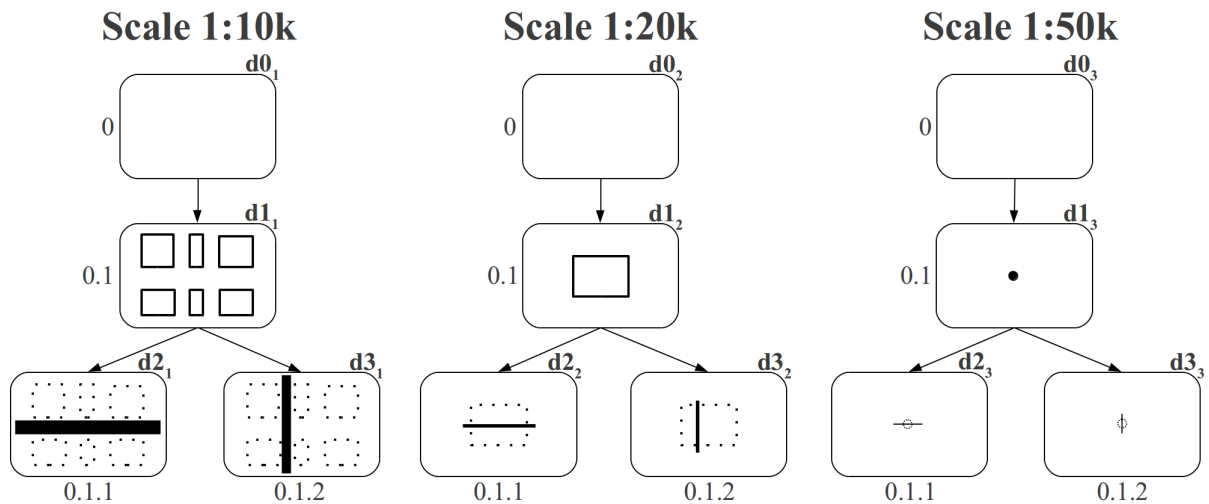

Figure 3.3: Multi-scale versioning problem example

Internal details appear in Figure 3.4. Part (a) shows the *multiversion objects*. Part (b) shows the *physical versions* and their geometry. Finally, the *logical versions* and their

relationship with *physical versions* are shown in part (c). For instance, in scale 1:10000, the city section is stored as a complex geometry (a polygon with six sub-polygons), with *oid o1* and with three physical representations – one per scale – *pv1*, *pv2* and *pv3*. Each of these geometries will be accessible via a different DBV, respectively $d1_1$, $d1_2$ and $d1_3$.

| (a) | (b) | (c) |

Multiversion Object			Logical Version		
Oid	**Title**		**DBV**	**Multiversion Object**	**Physical Version**
o1	city section		$d1_1$		pv1
o2	horizontal road		$d1_2$	o1	pv2
o3	vertical road		$d1_3$		pv3
			$d2_1$		pv4
			$d2_2$	o2	pv5
			$d2_3$		pv6
			$d3_1$		pv7
			$d3_2$	o3	pv8
			$d3_3$		pv9

(b) panel: pv1, pv2, pv3 / pv4, pv5, pv6 / pv7, pv8, pv9
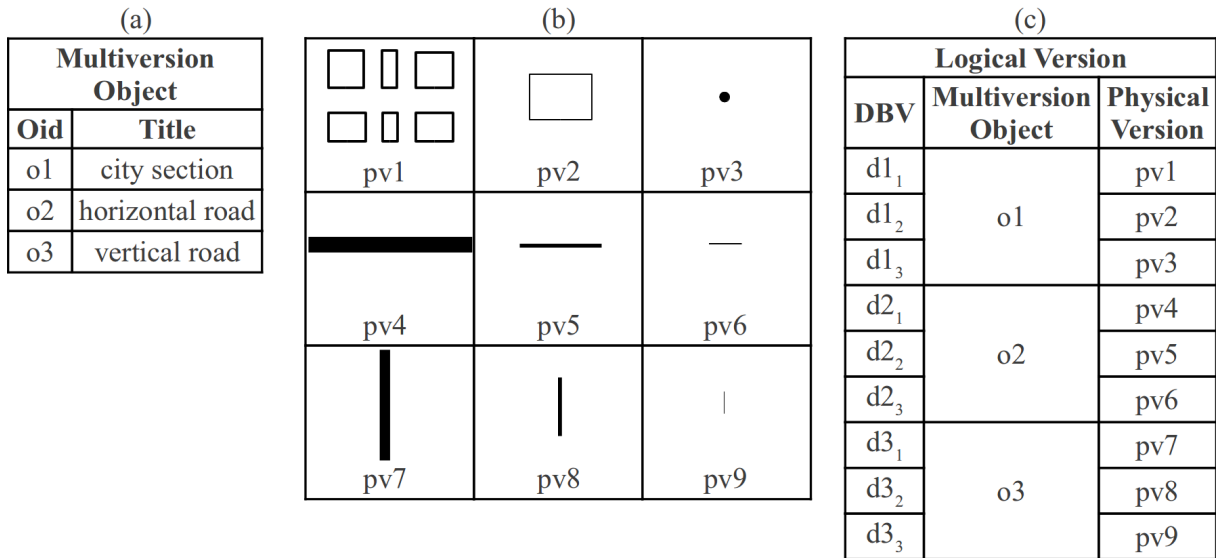
Figure 3.4: (a) *Multiversion objects*. (b) *Physical versions* and their geometry. (c) *Logical versions* from the example

Suppose the user wants to work at scale 1:10000, in the horizontal road situation, i.e., DBV $d2_1$. The DBV view is constructed from all objects explicitly assigned to it (*pv4* of *o2*), and all objects in previous DBVs of that scale, up to the root, i.e., $d1_1$ – *pv1* of *o1*. This construction of consistent scenarios for a given scale in time is achieved via the *stamps*, by the DBV mechanism. Notice that each version is stored only once. Unless objects change, their state is propagated through DBVs, saving space. Also, users can navigate across a path in the derivation tree, following the evolution of objects in time. For more details on space savings, see [8].

3.1.4 Aggregation

In multi-scale environments it is common for two or more objects to be aggregated in some scales. For this, a new object – an *aggregator* – must be created. An aggregator encapsulates n "children" objects in any scale, following the composition/aggregation principle of object-oriented databases.

Objects are aggregated in a scale s if they are not directly accessible in s, and they

are encapsulated within the same aggregator, which in turn is made available in *s*. We may also conclude that if an object is present in a scale, then it is not aggregated to any other in that scale.

As a simple example, consider three multiversion objects: *o1*, *o2* and *o3*, in which *o3* is an aggregator and represents the aggregation of *o1* and *o2*. Figure 3.5 illustrates this example and Figure 3.6 shows the tables related to *multiversion objects* (part (a)), aggregation (part (b)) and *logical versions* (part (c). We may note that in DBV $d1_1$, *o1* is represented by *pv1*, *o2* by *pv2* and *o3* does not appear. In turn, in DBV $d1_2$ just *o3* appears.
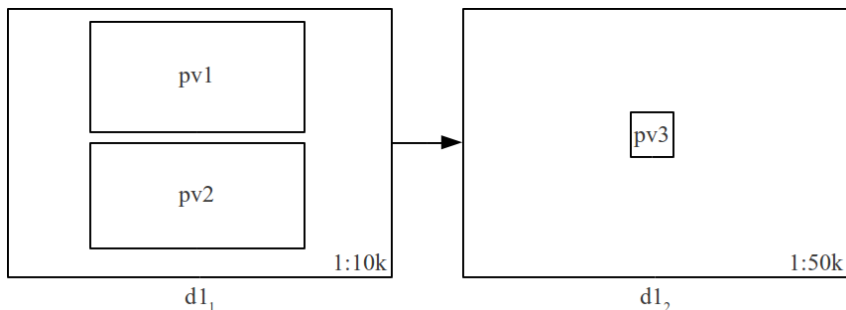
Figure 3.5: Example of aggregation using the DBV multi-scale model

(a)

Multiversion Object	
Oid	Title
o1	plot 1
o2	plot 2
o3	plots 1 and 2

(b)

Aggregation	
Aggregator	Encapsulated
o3	o1
o3	o2

(c)

Logical Version		
DBV	Multiversion Object	Physical Version
$d1_1$	o1	pv1
$d1_1$	o2	pv2
$d1_2$	o3	pv3

Figure 3.6: (a) *Multiversion objects.* (b) Aggregations. (c) *Logical versions*

3.2   Multi-scale integrity constraints

This section presents our set of primitive extensible integrity constraints (ICs) that define basic consistency conditions across scales. We point out that we are not interested in operations of generalization (e.g., McMaster and Shea's work [30]), but in multi-scale consistency upon updates.

We define a *multi-scale integrity constraint* (MS-IC) to be an IC that holds between consecutive scales s_i and s_j, where s_j represents the immediately smaller or larger scale related to s_i stored in the multiversion database. Constraints are to be applied within a *scenario*. Multi-scale consistency should be checked every time a new *scenario* is created and/or DBVs are updated (i.e., some multiversion object is inserted, deleted or modified). A *scenario* σ is consistent if its DBVs are consistent regarding MS-ICs.

Figure 3.7 shows 3 derivation trees, for scales *s1*, *s2* and *s3*, and exemplifies a scenario that contains all DBVs whose stamps are marked 0.1.2. MS-ICs are defined to hold between scales *s1* and *s2*, and *s2* and *s3*.
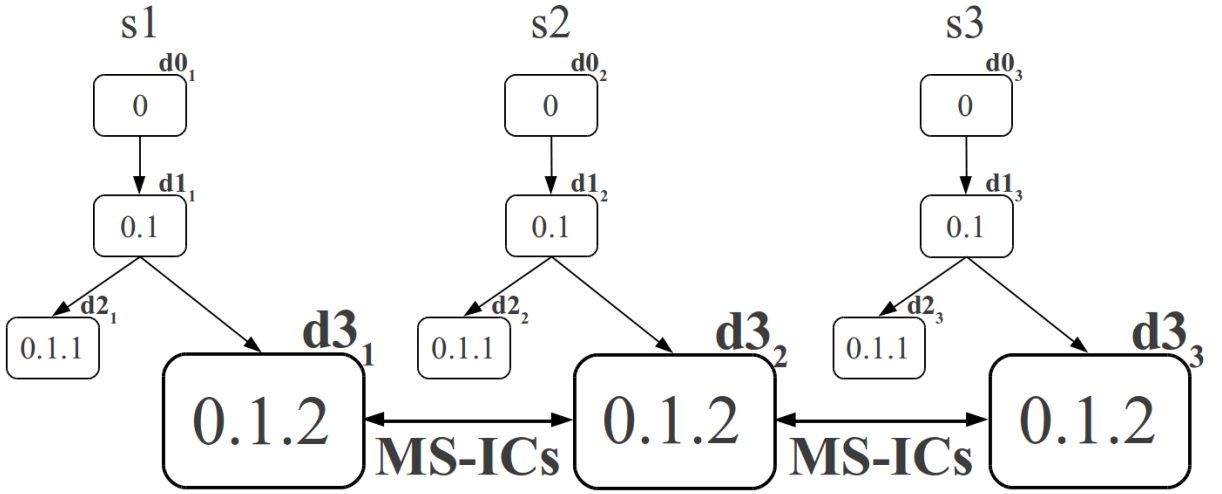
Figure 3.7: MS-ICs hold across consecutive scales

Definition. An MS-IC defines conditions under which two states of the world, represented in distinct spatial and/or temporal scales, are jointly consistent.

Let us consider a set of scales (spatial or temporal) $s_1 > s_2 > \ldots > s_n$ (s_1 is more detailed than s_2, s_2 is more detailed than s_3, and so on) and generic multiversion database objects, with schema <geom, {thematic_attr}, t_i, t_f>, where *geom* represents the geometry of the object specified by a set of coordinates, {*thematic_attr*} is its set of thematic attributes, t_i and t_f represent the initial and final validity timestamp, respectively, for that geometry and attributes.

There follow some operators defined by us to help define the MS-ICs:

- *simple_geom(multiversion_object$_o$, spatial_scale$_s$)* – returns *true* if the geometry component of the multiversion object o in s is point, line or polygon;

- *aggregator(multiversion_object$_o$)* – returns *true* if the multiversion object o is an

aggregator – an object created to encapsulate n "children" objects – e.g., a "city beach" object encapsulates one or more "house" objects;

- *num_points(multiversion_object$_o$, spatial_scale$_s$)* – returns the number of points that compose the geometry of o in s;

- *num_geometries(multiversion_object$_o$, spatial_scale$_s$)* – returns the number of geometries that compose the geometry of o in s;

- *children(multiversion_object$_o$)* – returns a set of multiversion objects that are encapsulated by o (i.e., it extracts the encapsulated objects, at the first encapsulation level);

- *aggregated(multiversion_object$_{o1}$, multiversion_object$_{o2}$, scale$_s$)* – returns *true* if multiversion objects *o1* and *o2* are aggregated in scale s (spatial or temporal) – e.g., if "house 1" and "house 2" in a scale *s1* are encapsulated by "block 1" in *s2 > s1*;

- *topR_applicable(topological_relationship$_t$, multiversion_object$_{o1}$, multiversion_object$_{o2}$, spatial_scale$_s$)* – returns *true* if the topological relationship t applies to the geometries of multiversion objects *o1* and *o2* in scale s. More details in subsection 3.2.1;

- *duration(multiversion_object$_o$, temporal_scale$_s$)* – computes $|t_i - t_f|$ of o in s;

- *interval(multiversion_object$_o$, temporal_scale$_s$)* – returns true if *duration(o, s) > 0*, i.e., it is an interval. Otherwise, it is an instant of time.

In the following subsections, we provide some examples for each type of MS-IC. Spatial and temporal scale variations are best separately represented as remarked by Camossi et al. [7]. Thus, we divide MS-ICs for spatial scale variation (Spatial MS-IC), temporal scale variation (Temporal MS-IC) and for one or the other (Shared MS-IC).

3.2.1 Spatial MS-ICs

Geometric IC

This category deals with the geometry of objects. Geometric integrity constraints are related to the *monotonicity assumption* – when there is a scale reduction, the geometry of objects must remain the same or be simplified [13, 20]. Thus, from the categorization of dimensions for 2D environment proposed by OGC [32] shown in Figure 3.8, we define the following MS-ICs for the DBV multi-scale model:

1. $\forall o \in s_i, s_j \ / \ s_i > s_j \Rightarrow \text{dimension}(o, s_i) \geq \text{dimension}(o, s_j)$

Geometry	Dimension
point	0
multi-point	0
line	1
multi-line	1
polygon	2
multi-polygon	2
geometry collection	the dimension of the component with the largest dimension

Figure 3.8: Geometry dimensions

2. $\forall o \in s_i, s_j \ / \ s_i > s_j \Rightarrow \text{num\_points}(o, s_i) \geq \text{num\_points}(o, s_j)$

3. $\forall o \in s_i, s_j \ / \ s_i > s_j \Rightarrow \text{num\_geometries}(o, s_i) \geq \text{num\_geometries}(o, s_j)$

4. $\forall o \in s_i, s_j \ / \ s_i < s_j \wedge \text{geometry\_collection}(o, s_i) \Rightarrow \text{geometry\_collection}(o, s_j)$

Defining only constraints on geometric dimensions (MS-IC 1) is not enough, because as shown in Figure 3.8, point and multi-point have the same dimension, and there is no consistency if in a larger scale an object is represented by a point and in a smaller scale the same object is represented by a multi-point geometry. The same applies to line and multi-line, polygon and multi-polygon, multi-polygon and collection, etc. MS-ICs 2 and 3 solve this issue, ensuring that the number of points and geometries will not increase when scale changes.

MS-IC 4 guarantees that if in a smaller scale an object is represented by a geometry collection, then in the immediate larger scale it should also be represented by a geometry collection.

Topological IC

This category is related to the topological relationships adopted by OGC [32]: *equals, disjoint, touches, crosses, within, overlaps, contains* and *intersects*. We do not consider *contains* and *intersects* relationships, because they are the inverse of *within* and *disjoint*, respectively.

Following OGC[32], let us consider three geometry groups: *P*, *L* and *A* (points, lines and areas), where *P* refers to 0-dimensional geometries, *L* to 1-dimensional geometries and *A* to 2-dimensional geometries. Figure 3.9 extracted from [32] shows what groups of relationships can be applied for each topological relationship. For instance, take the

topological relationship *overlaps* as an example. We can see in the Figure that the applicable groups of relationships are A/A, L/L and P/P – i.e., areas can overlap each other, and so can lines and points, but lines cannot overlap points.

Topological relationship	Applied groups of relationships
equals	all
disjoint	all
touches	all but P/P
crosses	P/L, P/A, L/L and L/A
within	all
overlaps	A/A, L/L and P/P

Figure 3.9: Topological relationships and applicable groups of relationships [32]

We state that topological relationships should be maintained from s_i to s_j if $s_i < s_j$. However, we do not define constraints for cases in which s_i is much smaller than s_j, because then anything can happen. The notion of "much smaller" is user and context-dependent, and must therefore be defined by the users. The implemented system supports this notion. For instance, in a precision farming situation a factor of 100 may be considered "much smaller", whereas in the context of global warming this may not be the case. *Topological ICs* deal with simple geometries, i.e., point, line and polygon, because for the other types (e.g. multi-polygon) it is dificult to define MS-ICs due to the variety of geometries which they may represent.

Given simple geometries *G1* and *G2*, and *R* a topological relationship between them, then *G1 R G2* in s_i ⇒ *G1 R G2* in s_j:

1. $\forall$(o1,o2) ∈ s_i,s_j, $\forall$R ∈ {equals, disjoint, touches, crosses, within, overlaps} / $s_i < s_j$ ∧ simple_geom(o1, s_i) ∧ simple_geom(o2, s_i) ∧ simple_geom(o1, s_j) ∧ simple_geom(o2, s_j) ∧ topR_applicable(o1, o2, s_i) ∧ topR_applicable(o1, o2, s_j) ∧ R(o1, o2, s_i) ⇒ R(o1, o2, s_j)

Directional IC

This category deals with the directional relations shown in Figure 3.10. The numbers in the Figure represent the direction relations regarding Qc – the reference point symbol [37]. For instance, when object *o* is in partition 1, then *o* is *north_west* of Qc.

We define that the direction between two disjoint objects in s_i and in s_j should be maintained across scales. For instance, if in s_i *o1* is *east* of *o2*, then in s_j the same should

occur. We also point out that more complex directional constraints are fuzzy, but may be defined by a combination of these constraints.
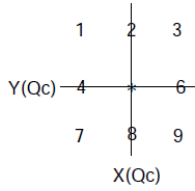
a. Plane partitions   **b. Primitive relations**   **c. Positions in the index**

1: north_west
2: restricted_north
3: north_east
4: restricted_west
5,*: same_position
6: restricted_east
7: south_west
8: restricted_south
9: south_east

Figure 3.10: Directional relations considering one point per object [37]

1. $\forall$(o1,o2) $\in$ s_i,s_j / $s_i > s_j \wedge$ simple_geom(o1, s_i) $\wedge$ simple_geom(o1, s_j) $\wedge$ simple_geom(o2, s_i) $\wedge$ simple_geom(o2, s_j) $\wedge$ disjoint(o1, o2, s_i) $\wedge$ disjoint(o1, o2, s_j) $\Rightarrow$ direction(o1, o2, s_i) = direction(o1, o2, s_j)

3.2.2 Temporal MS-ICs

Interval IC

This category deals with changes of time intervals across scales. Basically, larger scales should have longer or equal intervals than smaller scales. For instance, if some object is valid for a week in a week-based scale, it cannot be valid for a larger period in a year-based scale.

1. $\forall$o $\in$ s_i,s_j / $s_i > s_j \Rightarrow$ duration(o, s_i) $\geq$ duration(o, s_j)

Temporal Relationship IC

This category deals with temporal relations like: *before*, *meets*, *overlaps*, etc, as shown in Figure 3.11, which is based on Allen's theory [1].

Given time intervals *T1* and *T2*, and *R* a temporal relationship between them, then *T1 R T2* in $s_i \Rightarrow$ *T1 R T2* in s_j:

1. $\forall$(o1,o2) $\in$ s_i,s_j, $\forall$R $\in$ {before, equals, meets, overlaps, during, starts, finishes} / $s_i < s_j \wedge$ interval(o1, s_i) $\wedge$ interval(o2, s_i) $\wedge$ interval(o1, s_i) $\wedge$ interval(o2, s_j) $\wedge$ R(o1, o2, s_i) $\Rightarrow$ R(o1, o2, s_j)

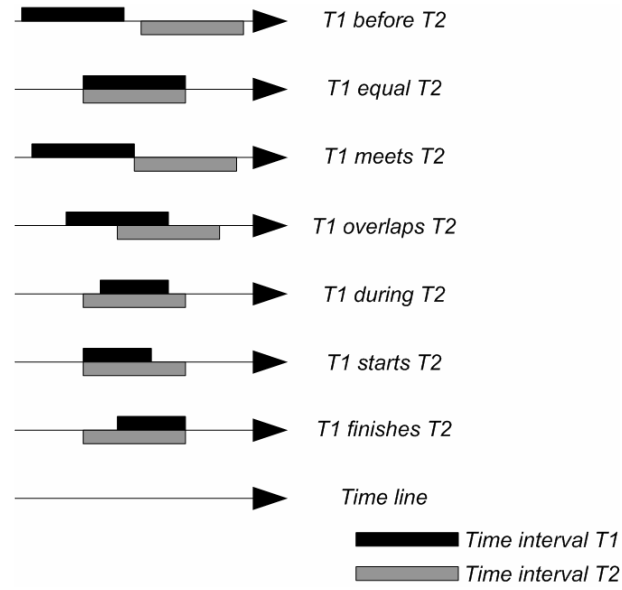Figure 3.11: Temporal relationships for intervals defined by Allen [1]. Adapted by [15]

3.2.3 Shared MS-ICs

Aggregation IC

MS-ICs related to aggregated objects fit in this category.

1. $\forall(o1,o2) \: / \: s_i > s_j \wedge \text{aggregated}(o1, o2, s_i) \Rightarrow \text{aggregated}(o1, o2, s_j)$

2. $\forall \text{aggregator}(o) \in s_i \wedge \notin s_j \: / \: s_i < s_j \Rightarrow \text{children}(o) \in s_j$

Existence IC

Consistency related to the existence or not of objects in s_i and s_j is set in this category.

1. $\forall o \in s_i \: / \: s_i < s_j \wedge \neg \text{aggregator}(o) \Rightarrow o \in s_j$

This MS-IC ensures that if an object exists in a smaller scale and it is not an aggregator, then it will also exist in the immediately larger scale.

Thematic IC

This category deals with thematic attributes of objects.

1. $\forall o \in s_i, s_j \Rightarrow \text{thematic}(o, s_i) = \text{thematic}(o, s_j)$

This constraint ensures that for every object *o* that appears in s_i and s_j, the values of its thematic attributes should be the same in s_i and s_j.

3.3   Conclusion

The first part of this Chapter presented the DBV multi-scale model, which is an extension of the DBV model that supports the management of multi-scale geospatial data. The second part defined some multi-scale integrity constraints to be applied to the DBV multi-scale model.

Chapter 4

Implementation details

4.1   DBV multi-scale platform

We developed our platform[1] on top of the PostGIS[2] spatial database extension for Post-greSQL[3] due to its widespread adoption and to its support of geospatial features. Our implementation uses the Java programming language, Java Persistence API (JPA)[4] and Generic Spatial DAO library[5], which is a generic DAO (Data Access Object)[6] with spatial extensions (using Hibernate Spatial[7]) and utility methods, for geographic data object/relational mapping.

Figure 4.1 shows the UML model of the platform, divided in five packages: *Domain Data Mapping*, *Database Handlers*, *Consistency Handlers*, *Multi-Scale Integrity Constraints* and *Controller*. The *Domain Data Mapping* package implements the database for the model of Figure 3.2, mapping Java objects into the underlying DBMS. The *Database Handlers* package uses the *Domain Data Mapping* classes to access the physical storage. The *Database Handlers* classes are inspired in the DAO pattern, to which we added specific methods of our model (e.g., to handle with logical versions). *Consistency Handlers* use the MS-ICs in the *Multi-Scale Integrity Constraints* package to check multi-scale consistency. Finally, the *Controller* package is accessed by applications to select the scenario to use and to perform operations on DBVs and objects.

The *Controller* plays the role of mediator among the platform's modules. For database operations, it receives query/update requests and invokes the appropriate methods of

---

[1]http://code.google.com/p/dbv-ms-platform
[2]http://postgis.refractions.net
[3]http://www.postgresql.org
[4]http://jcp.org/aboutJava/communityprocess/final/jsr317/index.html
[5]http://code.google.com/p/generic-spatial-dao
[6]http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html
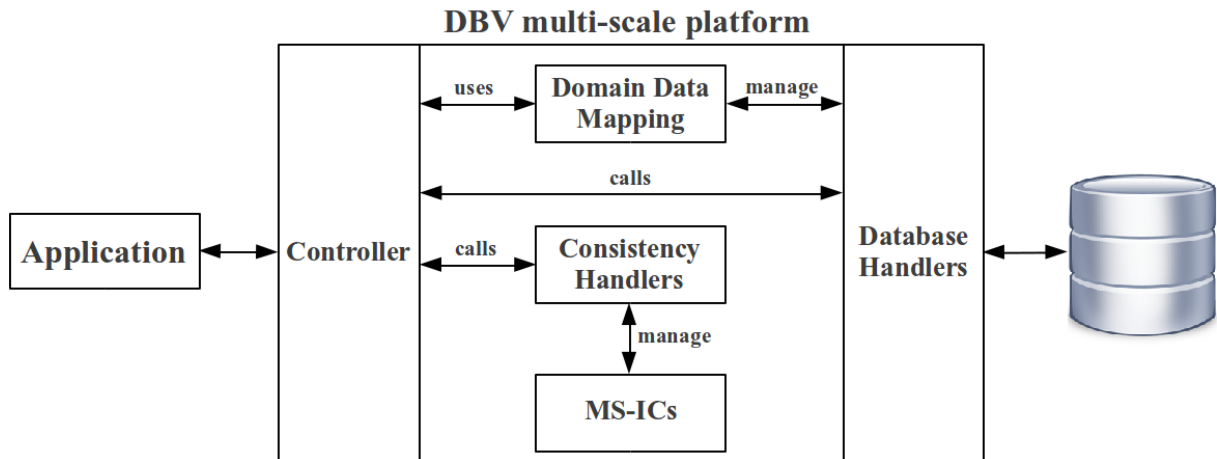[7]http://www.hibernatespatial.org

Figure 4.1: Architecture of the platform

package *Database Handlers* – e.g., create DBVs, scales, objects (physical and logical versions). Whenever a new scenario is created, or whenever the application requests a consistency check, it is the *Controller* that gets the necessary data from the database – by using the *Database Handlers*; it next invokes methods of the Consistency Handlers, which will in turn invoke different MS-IC methods. We recall that, moreover, the creation of a new scenario requires previously checking the consistency of the current scenario.

The basic default schema of the DBV multi-scale platform (limited to *GenericPV* subclass) is shown in Figure 4.2.

The *MultiversionObject* class has five attributes. The first is the identifier, the second is some title which identifies the object in the real world, the third is some complement of the *title* attribute, the fourth says if the object is an aggregator (more details in section 3.1.4), and the fifth is the set of encapsulated objects if this is an aggregator.

The *Scale* class has an identifier, an attribute that indicates the type of the scale (spatial, temporal, etc) and another that is the value associated to the type (e.g., "1:10000" for spatial scales, "minutes" for temporal scales).

The *DBV* class has five attributes. The first is the identifier, the second is the version *stamp*, the third is the associated scale, the fourth stores the *stamp* of the next child to be created by derivation (e.g., if a DBV with stamp 0.1 has already two children – 0.1.1 and 0.1.2 – the next child attribute will indicate 0.1.3), and the fifth is the DBV from which it was derived. The *PhysicalVersion* abstract class has an identifier, a geometry, and an initial and final timestamp: t_i and t_f, respectively. The *GenericPV* subclass has no aditional data besides *PhysicalVersion* attributes (users can create other subclasses of *PhysicalVersion* entering new attributes to be versioned). Finally, the *LogicalVersion*
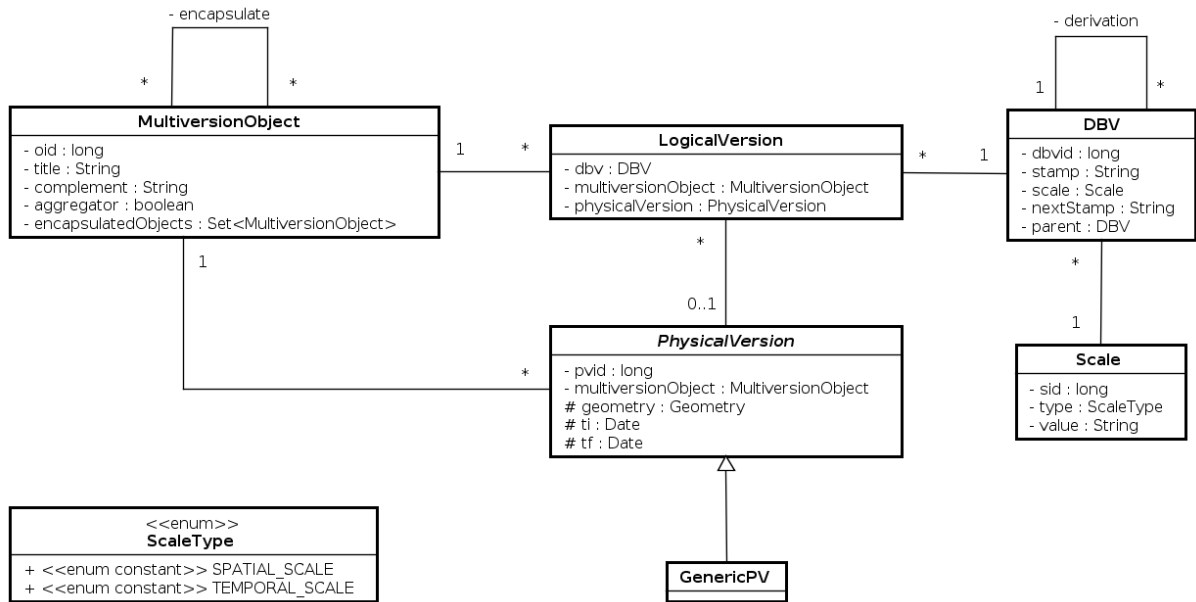
Figure 4.2: Entities of the platform described in UML

class links a DBV and a multiversion object with a *physical version.*

4.2 Platform functionality

This section describes details of some of the operations of the model: create a new scenario, accessing a DBV, adding or modifying a logical version of an object in a DBV, checking multi-scale consistency of the current scenario. Other operations are straightforward and are not described here. All operations involve read operations, which are not explicitly described here.

4.2.1 Creating a new scenario

A new scenario is created as follows. First, a current scenario *sc* must to be chosen. Then, for each DBV *d* in *sc*, the *Controller* interacts with the *DBVHandler* so that it creates a child for *d*. DBV creation can only happen if the scenario is consistent (see section 4.2.4).

4.2.2 Accessing a DBV

The access to DBV *X* of scale *s* is as follows. Suppose *X* = 0.1.1. The *Controller* delegates the task of accessing a DBV to the *LogicalVersionHandler*, which executes a query similar

to:

```
SELECT *
FROM logical_version, dbv, multiversion_object
WHERE logical_version.dbvid = dbv.dbvid AND
      logical_version.oid = multiversion_object.oid AND
      dbv.sid = 1 AND
      (dbv.stamp = '0.1.1' OR dbv.stamp = '0.1' OR dbv.stamp = '0')
ORDER BY dbv.stamp DESC
```

Basically, this is a join between *logical_version*, *multiversion_object* and *dbv* tables, looking for shared logical versions. Note that we reverse sort the result by the DBV *stamp*. This will cause the query to return first the multiversion objects of the nearest parent DBVs. In the implementation, we limit the maximum number of alternative scenarios to nine, but this can be extended arbitrarily, allowing the database to reverse sort correctly the stamps. For instance, in the present implementation, databases without modifications can reverse sort correctly (according to the DBV model) the stamps 0.9.1 and 0.8.1, but not 0.10.1 and 0.9.1.

Once the query is executed, we have a partial result: a list of logical versions in which multiversion objects may appear repeated. Next, we traverse this list putting the multiversion objects and their physical versions in a Java map, choosing always the first logical versions from those with repeated objects.

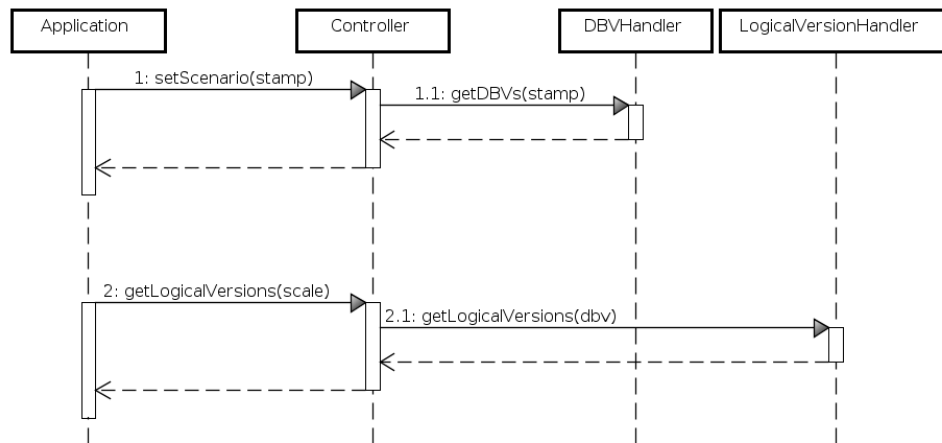Figure 4.3 shows the UML sequence diagram of the entire operation.

Figure 4.3: Accessing a DBV

Algorithm 1 Adding or modifying a logical version of an object

Require: a logical version object lv (representing a tuple $< dbvid, oid, pvid >$)

1: **procedure** ADDORMODIFY(lv)
2:     let lv_db and s_lv_db be logical version objects;
3:     $lv\_db \leftarrow$ search for a logical version object with the same primary key of lv;
4:     **if** $lv\_db <> null$ **then**   ▷ logical version already exists
5:         $lv\_db.pvid \leftarrow lv.pvid$
6:         modify tuple represented by lv_db
7:         **return**
8:     **end if**
9:     $s\_lv\_db \leftarrow$ search for a shared logical version of lv;
10:    **if** $s\_lv\_db == null$ **then**   ▷ shared logical version not found
11:        add the tuple represented by lv
12:    **else if** $s\_lv\_db.pvid <> lv.pvid$ **then**
13:        $s\_lv\_db.pvid \leftarrow lv.pvid$
14:        modify tuple represented by s_lv_db
15:    **end if**
16: **end procedure**

4.2.3 Adding or modifying a logical version of an object in a DBV

Users always start from a given scenario, which is the starting point of any operation.

Once the scenario is chosen, the *Controller* can receive a multiversion object, its OBJ-TYPE value and the target scale. Next, the *Controller* asks the *PhysicalVersionHandler* to store the OBJTYPE value as a physical version in the database.

Next, the *Controller* chooses the appropriate DBV in the current scenario, encapsulates this DBV in a *LogicalVersion* object lv along with the multiversion object and the physical version created. Finally, the lv object is passed to the *LogicalVersionHandler*, which executes the algorithm 1.

A shared logical version of an object o is the first logical version of o present in parent DBVs, following the data propagation of the DBV model.

Figure 4.4 shows the UML sequence diagram of the entire operation.

4.2.4 Checking multi-scale consistency of the current scenario

Consistency is only checked within a scenario, going through all scales, and checking all MS-ICs for every pair of scales s_i, s_{i+1}. Let us consider multi-scale consistency regarding spatial scales. Spatial MS-ICs are checked via two main methods present in
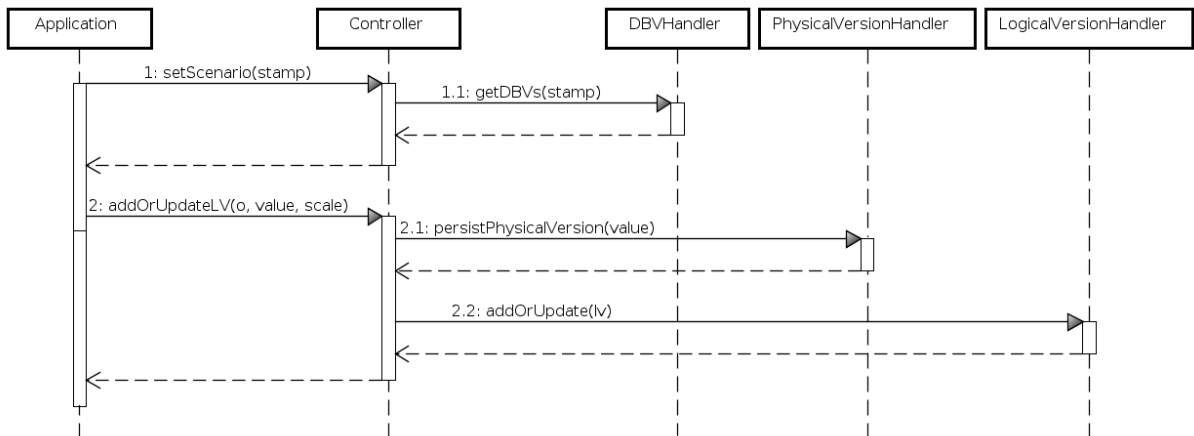
Figure 4.4: Adding or updating a logical version of an object

the classes that implement the interface *SpatialMSIC* (e.g., *GeometricIC* class): *checkConsistency()* and *checkPairConsistency()*. The *checkConsistency()* method checks the multi-scale consistency between two versions of an object, i.e., one version per scale. In turn, the *checkPairConsistency()* method checks the multi-scale consistency between two pairs of objects (one pair per scale). Algorithm 2 shows the checking of spatial multi-scale consistency.

This approach is not the best considering performance and memory and we leave for future work improvements on these issues.

Figure 4.5 shows the UML sequence diagram of this operation. Part of the execution of the algorithm 2 is executed by the *MultiScaleICHandler*.
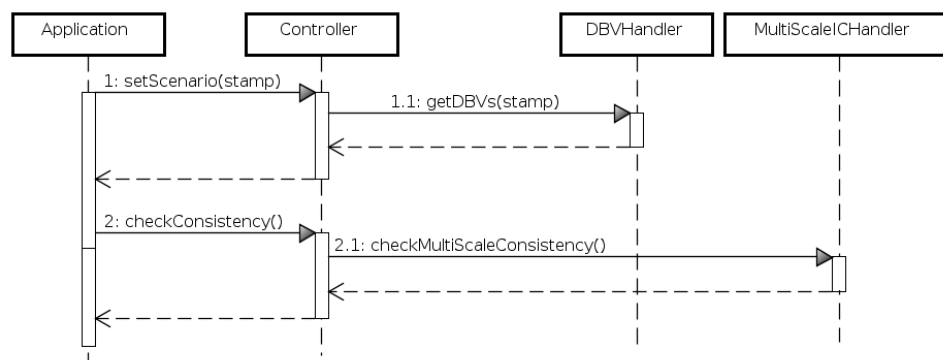

Figure 4.5: Checking multi-scale consistency of the current scenario

---

Algorithm 2 Checking spatial multi-scale consistency of the current scenario

Require: a scenario *sc* and a set of Spatial MS-ICs *spatial_msics*

1: **procedure** CHECKSPATIALMULTISCALECONSISTENCY(sc, spatial_msics)
2:     let *s* be the array of scales in *sc*;
3:     let *lvs_si* and *lvs_sj* be lists of logical versions at $s[i]$ and $s[j]$, where $j = i + 1$;
4:     let *geom_o1_si*, *geom_o1_sj*, *geom_o2_si* and *geom_o2_sj* be geometries;
5:     **for** $i \leftarrow 1$ **to** *s.length* **do**
6:         $j \leftarrow i + 1$;
7:         *lvs_si* ← get logical versions (sorted by the title of the objects) in $s[i]$;
8:         *lvs_sj* ← get logical versions (sorted by the title of the objects) in $s[j]$;
9:         **for** $k \leftarrow 1$ **to** *lvs_si.length* **do**
10:             *geom_o1_si* ← get the geometry contained in *lvs_si*[k];
11:             *geom_o1_sj* ← get the geometry contained in *lvs_sj*[k];
12:             **for all** constraints in *spatial_msics* **do**
13:                 check the consistency of the pair $< geom\_o1\_si, geom\_o1\_sj >$;
14:             **end for**
15:             **for** $w \leftarrow k + 1$ **to** *lvs_si.length* **do**
16:                 *geom_o2_si* ← get the geometry contained in *lvs_si*[w];
17:                 *geom_o2_sj* ← get the geometry contained in *lvs_sj*[w];
18:                 **for all** constraints in *spatial_msics* **do**
19:                     check the consistency between the two pairs
20: $< geom\_o1\_si, geom\_o2\_si >$ and $< geom\_o1\_sj, geom\_o2\_sj >$;
21:                 **end for**
22:             **end for**
23:         **end for**
24:     **end for**
25: **end procedure**

4.3   DBV multi-scale web manager

In order to visualize the stored multi-scale data and check their consistency in a straight-forward way, we developed a web application called "DBV multi-scale web manager". It was built using the Java programming language, JSF[8] (Java Server Faces) framework, RichFaces[9] visual components for JSF, OL4JSF[10] library, which helps the use of Open-Layers[11] for JSF. This application can be seen as a high level layer between the user and the DBV multi-scale platform, as shown in Figure 4.6. Any user request through this

---

[8]http://javaserverfaces.java.net
[9]http://www.jboss.org/richfaces
[10]http://java.net/projects/ol4jsf
[11]http://openlayers.org

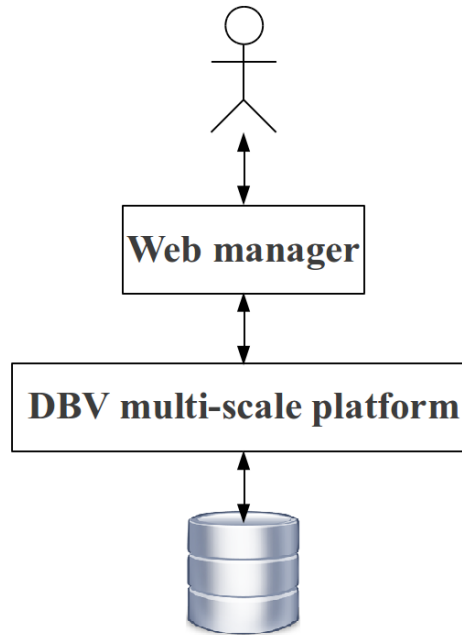manager invokes commands of the DBV multi-scale platform.

Figure 4.6: DBV multi-scale web manager

Figure 4.7 shows a screen copy of the interface of the DBV multi-scale web manager. This Figure concerns the simple example of section 3.1.3. The menu (left side) containing the scenarios is built dynamically by looking at the DBVs available in the underlying database. Also, there are three tabs above the map. The first allows to see the map, the second shows the multi-scale data in a table and the third allows to check multi-scale consistency. The map window has some basic controls like zoom, navigation, etc, provided by the OpenLayers library. The combo box at the top allows users to alternate across persistence units. Each persistence unit configures access to a different multiversion database. The Figure shows that the current persistence unit is *dbv-ms-platform-example*.

4.4   Conclusion

This Chapter presented some implementation details of the DBV multi-scale platform. The next Chapter presents a case study using real data.
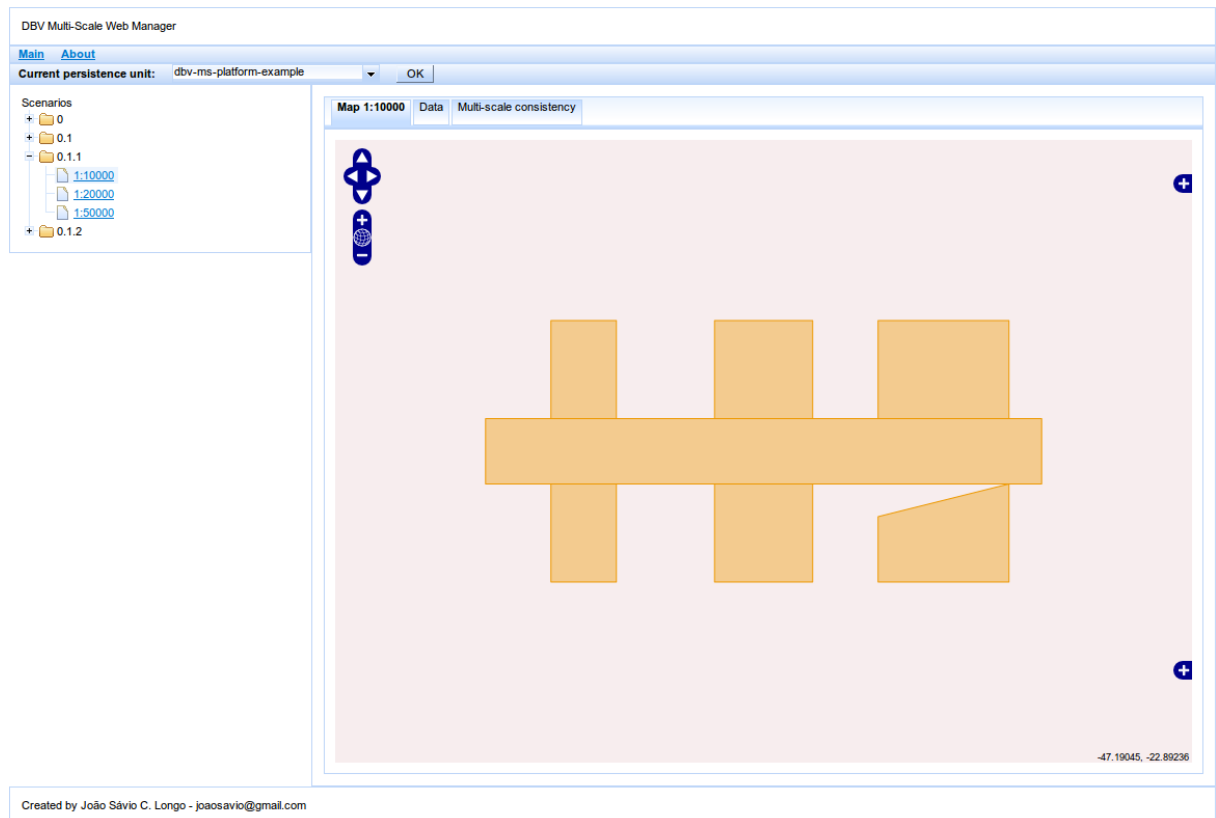
Figure 4.7: Screen copy of the interface of the DBV multi-scale web manager

Chapter 5

Case study

5.1 User interactions

This chapter gives an example of a user session. This case study used multi-scale geospatial real world data provided by Embrapa[1]. The data sets contain 5641 geometry features related to the Rio Pardo watershed and its rivers, at two scales: 1:250k and 1:1M.

Let us consider the user wants to construct some scenarios using the DBV multi-scale platform. These scenarios are described in Figure 5.1.

Scenario	Multi-scale data
0	
0.1	- watershed polygons
0.1.1	- watershed polygons - all rivers
0.1.2	- watershed polygons - main river

Figure 5.1: Proposed scenarios

The user starts by choosing to create empty DBVs, one for each scale. Figure 5.2 shows screen copies of this first step, one per scale. The left side of each screen shows how new versions are progressively created, while the maps portray the actual visualization of multi-scale data at each DBV.

Next, consider that the user wants to show more data details reflecting evolution in the user's knowledge of the world. This is achieved by demanding the creation of two new
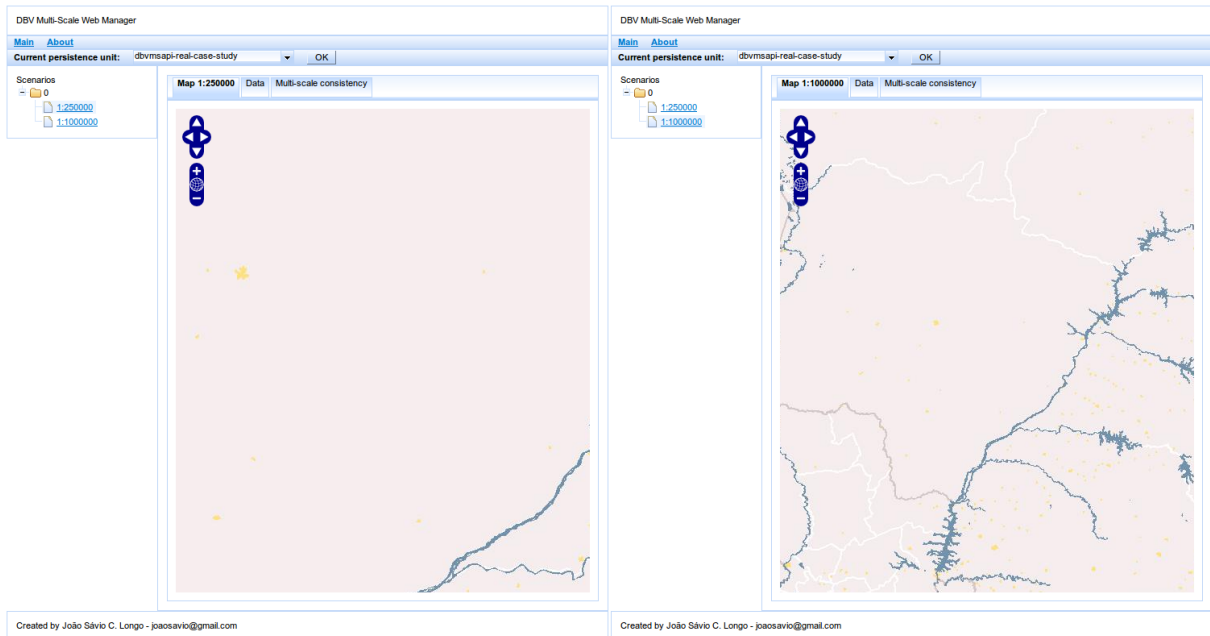
---

[1]http://www.embrapa.br

Figure 5.2: Scenario 0

DBVs (one for each scale) descending from scenario 0. Figure 5.3 shows the scenarios for DBVs with stamp 0.1.

Next, suppose the user wants to add data in order to represent two alternative scenarios: one contains the watershed and all its rivers and another with the watershed and only its main river. For each scale, two new DBVs are created descending from scenario 0.1. Figures 5.4 and 5.5 are screen copies of these two alternative scenarios, respectively numbered 0.1.1 and 0.1.2.

At each DBV and scale, the user can visualize more or less details by clicking on the +/- buttons. Figures 5.6 shows an example of zooming into the screen of scenario 0.1.1, for scale 1:250000.

5.2 Underlying implementation issues

Figure 5.7 shows the derivation trees of the case study. Scenario 0 has no data (by default), scenario 0.1 has only the watershed polygons in the two scales. Scenario 0.1.1 has the watershed with all rivers in the two scales. Here, we do not have to store the watershed polygons again, because they are shared from the previous scenario. The same occurs in scenario 0.1.2, where besides the watershed polygons, the main river appears.

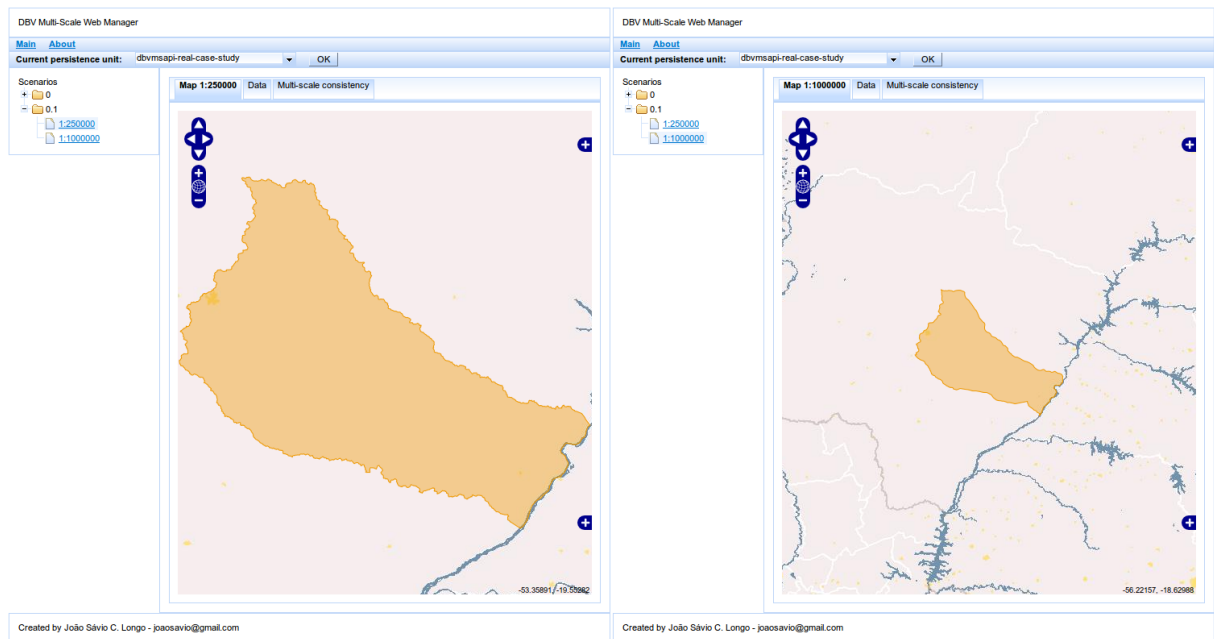The DBV multi-scale platform was used to store these versions in the database. First
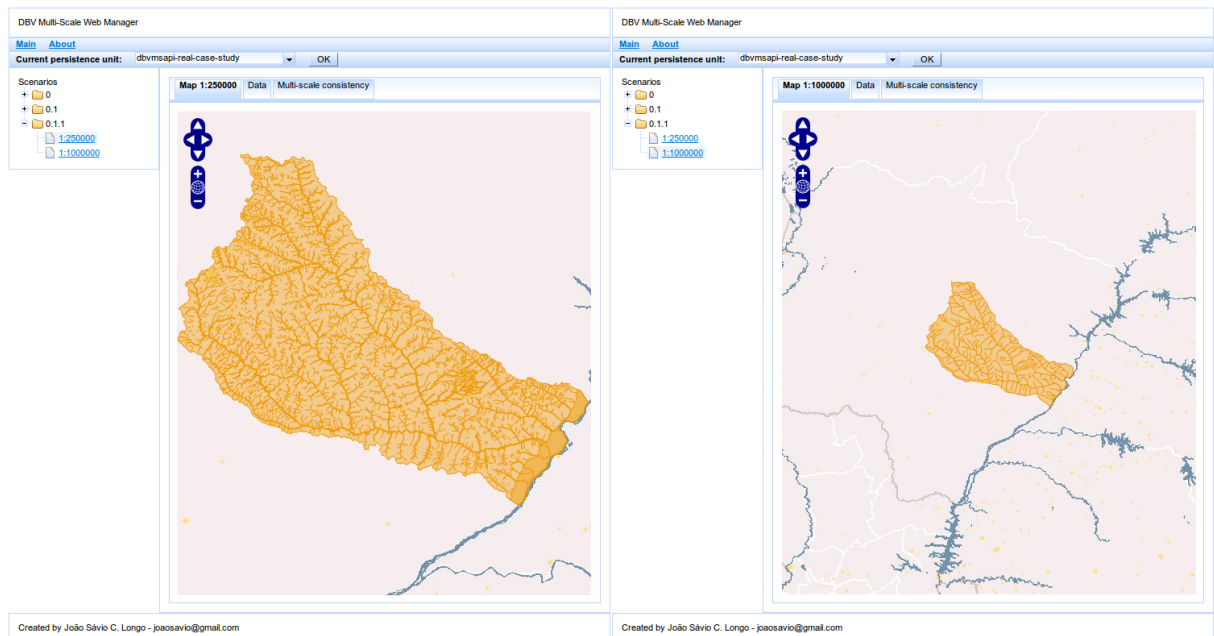
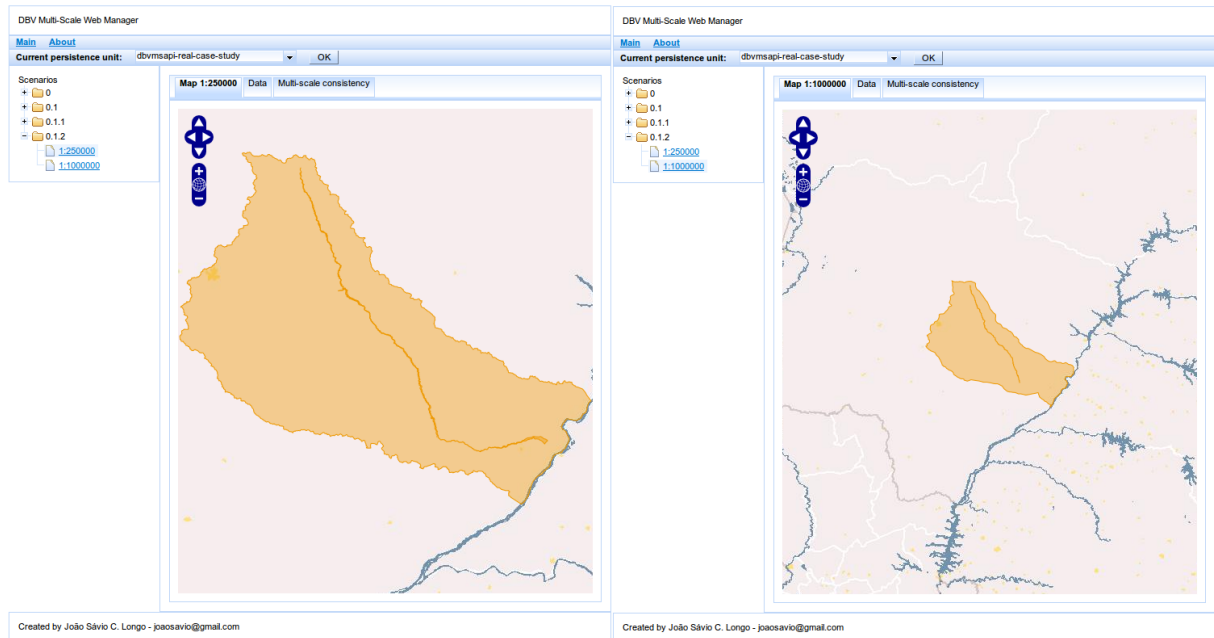Figure 5.3: Scenario 0.1

Figure 5.4: Scenario 0.1.1

Figure 5.5: Scenario 0.1.2
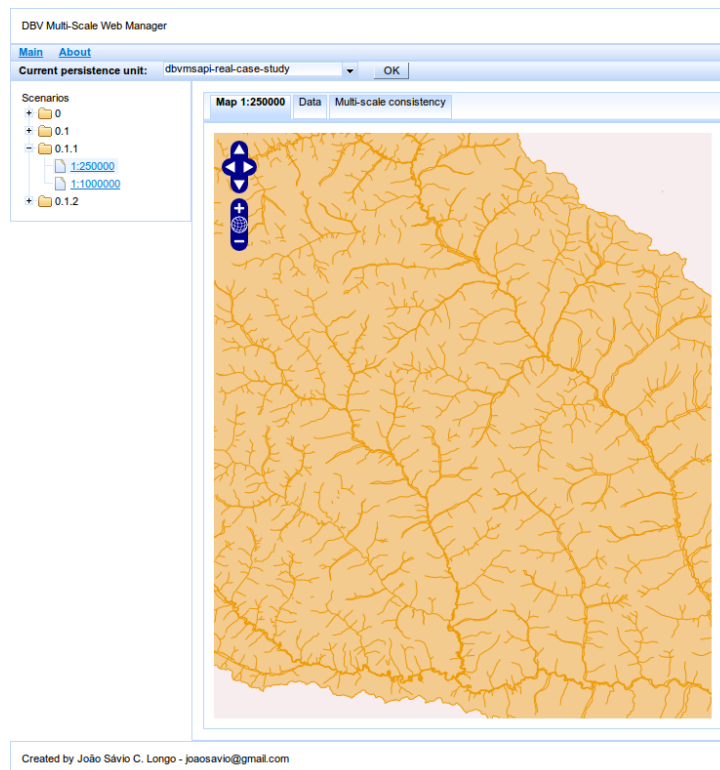
Figure 5.6: Scenario 0.1.1 with more details
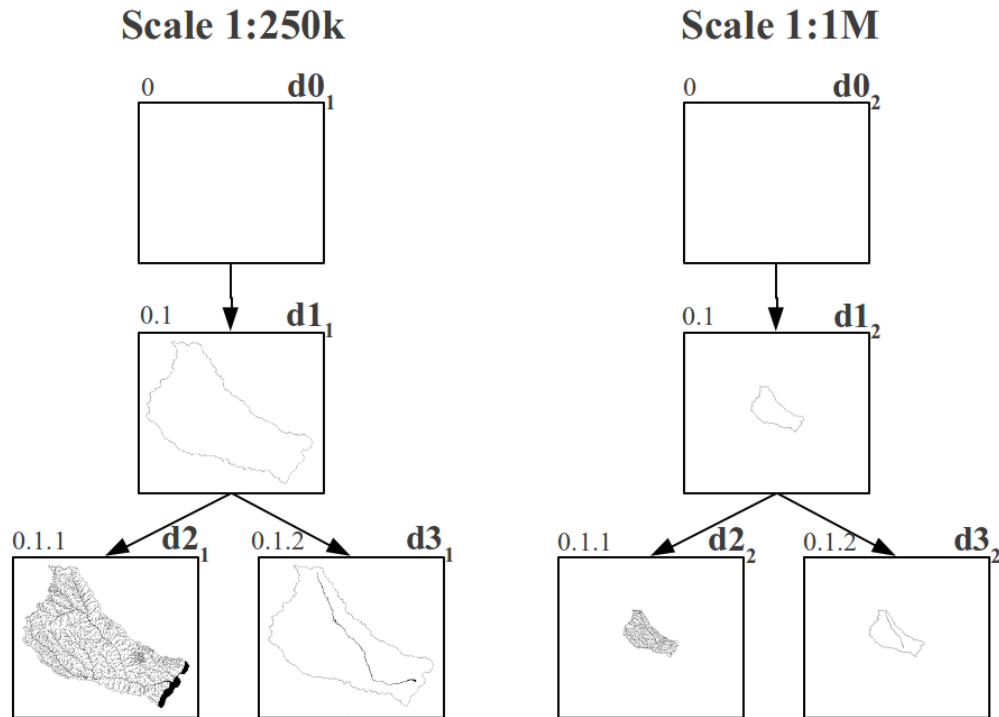
Figure 5.7: Derivation trees of the case study

of all, we chose to use the default schema, i.e., the *GenericPV* subclass of *PhysicalVersion*. Next, we defined that two scales have to be created: scales 1:250k and 1:1M. Every time a scale is added, a new derivation tree is created – by updating a root DBV.

Scenario 0.1 was constructed by creation of a DBV per scale, followed by insertion of objects corresponding to watershed and rivers. This was achieved by invoking operations on multiversion objects and DBVs, via invocation of methods of the platform, e.g., adding or updating *logical versions* of objects, by working in a scale at a time. When a new DBV is created from the current, the changes are saved. Subsequent versions were built by changing the *logical versions* and creating new DBVs.

5.3   Experiments

We now discuss a few experiments performed in order to check some of the MS-ICs defined in Section 3.2.

5.3.1   Experiment 1

The database is supposed to be initially consistent (i.e., all scenarios are consistent with respect to the original data). The first experiment went through all scenarios, top-down in the trees, checking the consistency of all scenarios (i.e., all DBVs, all scales). This is done by clicking on the right button of Figure 5.8. The Figure shows the message presented to the user as a result of the consistency analysis, indicating that the entire database is consistent (for all scales analyzed).
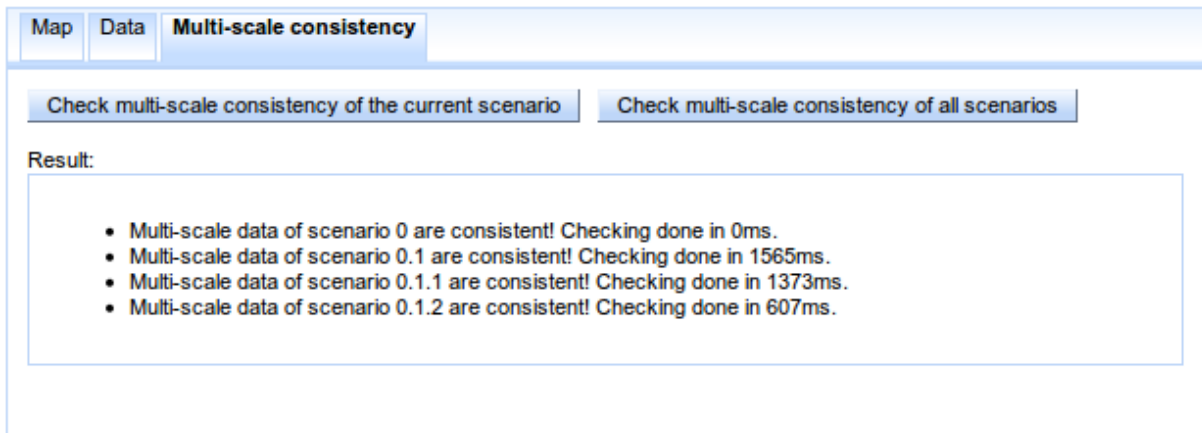
Figure 5.8: All scenarios are consistent in experiment 1

We point out that the times displayed indicate a better performance in scenarios 0.1.1 and 0.1.2 (both with many data) than scenario 0.1 (with few data). This is because the previous check data was already in the memory cache. This does not reflect the performance considerations regarding the DBV model, that concerns space savings, but not time.

5.3.2   Experiment 2

Consider that a user is updating the geometry of the main river in scenario 0.1.2. Suppose (s)he has inserted a simplified geometry in the largest scale and a detailed geometry in the smallest scale. Then, when the user requests a consistency check of the scenario, the system will indicate violation of constraints. Figure 5.9 shows a screen copy containing the messages to this effect.

Notice that several error messages appear, considering object dimensions, number of geometries and number of points of an object.
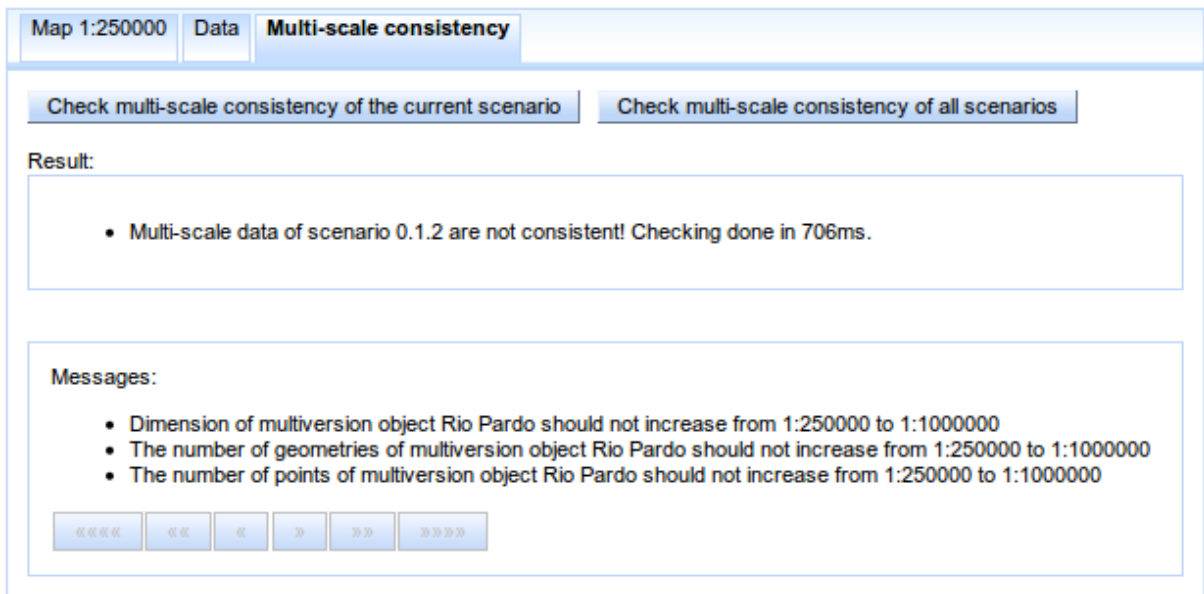
Figure 5.9: Consistency error messages of experiment 2

5.3.3 Experiment 3

Let us consider scenario 0.1.1. Suppose a given user inserted river *Rio Anhanduí* object in the smallest scale, but no corresponding object was inserted in the largest scale. Then, when (s)he checks the consistency of the scenario, the system will show one constraint error message – see Figure 5.10.

These kinds of check verify that scales are mutually consistent with respect to objects stored.

5.4 Conclusion

This Chapter presented an example of a user session, and outlined how this is implemented internally. The next Chapter presents conclusions and future directions.

Figure 5.10: Consistency error message of experiment 3

Chapter 6

Conclusions and extensions

6.1 Conclusions

We have presented an approach named DBV multi-scale model to manage multi-scale geospatial data, and keep track of their evolution. Our model supports the traceability of the evolution of spatial objects, while at the same time handling multi-scale data management. Thanks to the adoption of the DBV model as a basis, storage space is saved [8]. The separation between *physical* and *logic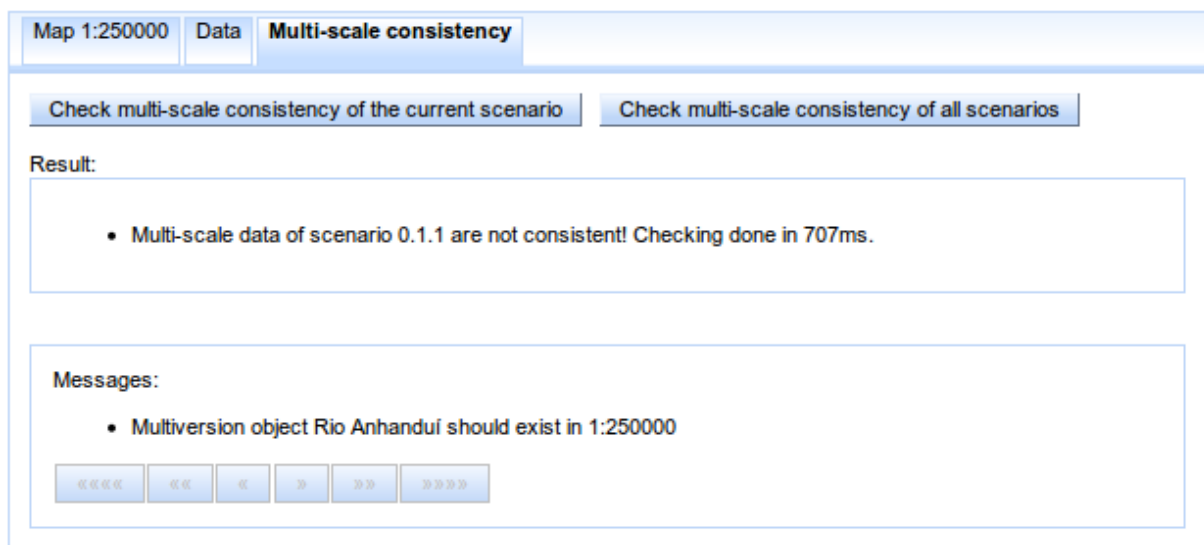al versions* facilitates the creation of consistent, single scale views over multi-scale data. We point out that our approach is centered on data structures to store and manage multi-scale data. This allows controlling updates, keeping history of evolution in the real world and other issues that can be efficiently handled only in a storage based policy. Nevertheless, the proposal can be used as a basis for any kind of generalization approach – e.g., construction of intermediate scales, generalization of alternative virtual scenarios, and so on, to work, for instance in digital cartography. Moreover, part of this work was also published in [27].

Regarding consistency, some multi-scale integrity constraints (MS-ICs) were proposed. We have defined MS-ICs for spatial and temporal scales, as well as generic multi-scale constraints.

The DBV multi-scale model was implemented in a prototype of a platform, developed in order to validate our solution. The prototype integrates the model with support to multi-scale integrity constraints. Finally, we tested this platform with multi-scale real data, thereby validating the proposal.

Recapping, considering the questions posed in the introduction, the contributions answer those questions as follows:

- Management of multi-scale data

  - How to manage multi-scale data?

DBV multi-scale model and platform

– How to trace real world evolution?
Multiple derivation trees

• Multi-scale consistency

– How to guarantee multi-scale consistency?
MS-ICs in the DBV multi-scale platform

6.2 Extensions

There are many extensions possible for this work. They include:

• add multi-scale constraints that combine the variation of the spatial scale along with the temporal scale, i.e., spatio-temporal constraints;

• add intra-scale constraints;

• support automatic update propagation across scales;

• add full support to multi-representation, in which each representation would correspond to a different derivation tree and there would be multi-representation constraints;

• improve consistency checking algorithms;

• add semantics to multiversion objects.

Also, the DBV multi-scale web manager could be extended to allow full interaction with the DBV multi-scale platform. At present, it is only possible to visualize stored multi-scale data and check consistency. Our implementation considered consistency checking via software (as opposed to active databases), which extensibility and does not require active mechanisms. A possible additional extension would be to consider this implementation alternative.

Bibliography

[1] J. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[2] E. Bertino, E. Camossi, and M. Bertolotto. Multi-granular spatio-temporal object models: Concepts and research directions. In *Object Databases*, volume 5936, pages 132–148. Springer Berlin / Heidelberg, 2010.

[3] M. Bobzien, D. Burghardt, I. Petzold, M. Neun, and R. Weibel. Multi-representation databases with explicitly modeled horizontal, vertical, and update relations. *Cartography and Geographic Information Science*, 35(1):3–16, 2008.

[4] D. Burghardt, I. Petzold, and M. Bobzien. Relation modelling within multiple representation databases and generalisation services. *Cartographic Journal, The*, 47(3):238–249, 2010.

[5] Y. Bédard, E. Bernier, and T. Badard. Multiple representation spatial databases and the concept of vuel. *Encyclopaedia in Geoinformatics, Hershey: Idea Group Publishing*, 2007.

[6] E. Camossi, E. Bertino, G. Guerrini, and M. Bertolotto. Adaptive management of multigranular spatio-temporal object attributes. In *Advances in Spatial and Temporal Databases*, volume 5644 of *Lecture Notes in Computer Science*, pages 320–337. Springer Berlin / Heidelberg, 2009.

[7] E. Camossi, M. Bertolotto, and E. Bertino. Multigranular spatio-temporal models: implementation challenges. In *Proc. of the 16th ACM SIGSPATIAL*, GIS '08, pages 63:1–63:4. ACM, 2008.

[8] W. Cellary and G. Jomier. Consistency of versions in object-oriented databases. In *Proc. of 16th International Conference on Very Large Databases*, pages 432–441. Morgan Kaufmann, 1990.

[9] S. Cockcroft. A taxonomy of spatial data integrity constraints. *GeoInformatica*, 1(4):327–343, 1997.

[10] S. Cockcroft. The design and implementation of a repository for the management of spatial data integrity constraints. *GeoInformatica*, 8:49–69, 2004.

[11] F. Currim, S. Currim, C. Dyreson, R. Snodgrass, S. Thomas, and R. Zhang. Adding temporal constraints to xml schema. *Knowledge and Data Engineering, IEEE Transactions on*, PP(99):1, 2011.

[12] F. Currim and S. Ram. Modeling spatial and temporal set-based constraints during conceptual database design. *Info. Sys. Research*, 23(1):109–128, 2012.

[13] V. Delis and T. Hadzilacos. On the assessment of generalisation consistency. In *Advances in Spatial Databases*, volume 1262 of *Lecture Notes in Computer Science*, pages 321–335. Springer Berlin / Heidelberg, 1997.

[14] X. Deng, H. Wu, and D. Li. Mrdb approach for geospatial data revision. In *Proc. of SPIE, the International Society for Optical Engineering*. Society of Photo-Optical Instrumentation Engineers, 2008.

[15] T. Dias, G. Câmara, and C. Davis. *Geographic Databases*, chapter Spatio-temporal models, pages 147–180. MundoGEO, 2005.

[16] A. Doucet, M. Fauvet, S. Gançarski, G Jomier, and S. Monties. Using database versions to implement temporal integrity constraints. In *CDB*, pages 219–233, 1997.

[17] A. Doucet, S. Gançarski, G. Jomier, and S Monties. Integrity constraints and versions. In *FMLDO*, pages 25–39, 1996.

[18] A. Doucet, S. Gançarski, G. Jomier, and S. Monties. Integrity constraints in multiversion databases. In *BNCOD*, pages 56–73, 1996.

[19] A. Doucet and S. Monties. Versions of integrity constraints in multiversion databases. In *DEXA*, pages 252–261, 1997.

[20] M. Egenhofer, E. Clementini, and P. Di Felice. Evaluating inconsistencies among multiple representations. In *Proc. of the Sixth International Symposium on Spatial Data Handling*, volume 2, pages 901–920, 1994.

[21] A. Friis-Christensen and C. Jensen. Object-relational management of multiply represented geographic entities. In *Proc. 15th International Conference on Scientific and Statistical Database Management SSDBM*, 2003.

[22] S. Gançarski and G. Jomier. A framework for programming multiversion databases. *Data Knowl. Eng.*, 36:29–53, January 2001.

[23] H. Gao, H. Zhang, D. Hu, R. Tian, and D. Guo. Multi-scale features of urban planning spatial data. In *18th International Conference on Geoinformatics*, pages 1–7, june 2010.

[24] D. Gubiani and A. Montanari. A conceptual spatial model supporting topologically-consistent multiple representations. In *Proc. of the 16th ACM SIGSPATIAL*, GIS '08, pages 9:1–9:10. ACM, 2008.

[25] R. Guting. An Introduction to Spatial Database Systems. *The VLDB Journal*, 3(4):357–400, 1994.

[26] M. Hampe, K. Anders, and M. Sester. Mrdb applications for data revision and real-time generalisation. In *Proceedings of the 21st International Cartographic Conference*, pages 10–16. Citeseer, 2003.

[27] J. Longo, L. Camargo, C. Medeiros, and A. Santanchè. Using the dbv model to maintain versions of multi-scale geospatial data. In *Advances in Conceptual Modeling*, volume 7518 of *Lecture Notes in Computer Science*, pages 284–293. Springer Berlin Heidelberg, 2012.

[28] S. Mäs and W. Reinhardt. Categories of geospatial and temporal integrity constraints. In *Advanced Geographic Information Systems Web Services, 2009. GEOWS '09. International Conference on*, pages 146–151, feb. 2009.

[29] S. Mäs, F. Wang, and W. Reinhardt. Using ontologies for integrity constraint definition. In *Proc. of the 4th International Symposium On Spatial Data Quality, ISSDQ*, pages 304–313, 2005.

[30] R. McMaster and K. Shea. Generalization in digital cartography. In *Resource Publication of the Association of American Geographers*, 1992.

[31] C. Medeiros and M. Cilia. Maintenance of binary topological constraints through active databases. In *Proceedings of the 3rd ACM Workshop on Advances in GIS*, pages 127–134. Citeseer, December 1995.

[32] OGC. Opengis implementation specification for geographic information - simple feature access - part 1: Common architecture. http://www.opengeospatial.org/standards/sfa.

[33] OMG. Ocl 2.3.1. http://www.omg.org/spec/OCL/2.3.1.

[34] P. Oosterom. Research and development in geo-information generalisation and multiple representation. *Computers, Environment and Urban Systems*, 33(5):303–310, 2009.

[35] P. Oosterom and J. Stoter. 5d data modelling: Full integration of 2d/3d space, time and scale dimensions. In *Proc. GIScience 2010*, pages 310–324, 2010.

[36] R. Pandey. Architectural description languages (adls) vs uml: a review. *SIGSOFT Softw. Eng. Notes*, 35:1–5, 2010.

[37] D. Papadias and T. Sellis. On the qualitative representation of spatial knowledge in 2d space. *VLDB Journal*, 3:479–516, 1994.

[38] C. Parent, S. Spaccapietra, C. Vangenot, and E. Zimányi. Multiple representation modeling. In *Encyclopedia of Database Systems*, pages 1844–1849. Springer US, 2009.

[39] C. Parent, S. Spaccapietra, and E. Zimányi. The murmur project: Modeling and querying multi-representation spatio-temporal databases. *Information Systems*, 31(8):733–769, 2006.

[40] Y. Piao, X. Wang, and Z. Shang. Integrated consistency constraints checking in a complicated development environment. In *International Symposium on Computational Intelligence and Design, ISCID '08*, volume 2, pages 190 –193, 2008.

[41] A. Ruas and C. Duchêne. Chapter 14 - a prototype generalisation system based on the multi-agent system paradigm. In *Generalisation of Geographic Information*, pages 269–284. Elsevier Science B.V., 2007.

[42] M. Salehi, Y. Bédard, M Mostafavi, and J. Brodeur. From transactional spatial databases integrity constraints to spatial datacubes integrity constraints. In *Proc. of the 5th International Symposium on Spatial Data Quality*, 2007.

[43] M. Salehi, Y. Bédard, M. Mostafavi, and J. Brodeur. Formal classification of integrity constraints in spatiotemporal database applications. *Journal of Visual Languages & Computing*, (5):323–339, 2011.

[44] L. Sarjakoski. Conceptual models of generalisation and multiple representation. In *Generalisation of Geographic Information*, pages 11–35. Elsevier Science B.V., 2007.

[45] S. Spaccapietra, C. Parent, and C. Vangenot. Gis databases: From multiscale to multirepresentation. In *Abstraction, Reformulation, and Approximation*, volume 1864 of *Lecture Notes in Computer Science*, pages 57–70. Springer Berlin / Heidelberg, 2000.

[46] J. Stoter, T. Visser, P. van Oosterom, W. Quak, and N. Bakker. A semantic-rich multi-scale information model for topography. *International Journal of Geographical Information Science*, 25(5):739–763, 2011.

[47] B. Thalheim. Integrity constraints in (conceptual) database models. In *The Evolution of Conceptual Modeling*, volume 6520 of *Lecture Notes in Computer Science*, pages 42–67. Springer Berlin / Heidelberg, 2011.

[48] F. Wang and W. Reinhardt. Extending geographic data modeling by adopting constraint decision table to specify spatial integrity constraints. In *The European Information Society*, Lecture Notes in Geoinformation and Cartography, pages 435–454. Springer Berlin Heidelberg, 2007.

[49] S. Zhou and C. Jones. A multirepresentation spatial data model. In *Proc. 8th International Symposium in Advances in Spatial and Temporal Databases – SSTD*, pages 394–411, 2003. LNCS 2750.