From the Conceptual Design of Spatial Constraints to their Implementation in Real Systems

Giuseppe Pelagatti, Mauro Negri
Dipartimento di Elettronica e Informazione,
Politecnico di Milano
Milan (Italy)
giuseppe.pelagatti@polimi.it

Alberto Belussi, Sara Migliorini
Dipartimento di Informatica
Università di Verona
Verona (Italy)
alberto.belussi@univr.it

ABSTRACT
The Spatial data community recognizes the need for procedures that automatically validate spatial integrity constraints defined at conceptual level. This validation becomes particularly important in an open and distributed environment, like a Spatial Data Infrastructure (SDI), where the level of integration and interoperability is very high. However, the current gap between the conceptual design of a spatial database and its implementation into a GIS system makes the definition of spatial constraints useful only for documentation purposes and not for automatic validation. In particular, the Standard ISO TC211 recommends the use of the OCL language for specifying spatial constraints at conceptual level: unfortunately this language is not so easy to use and understand by GIS users and no tool is available for automatically checking generic OCL constraints on datasets which should be conformant with the conceptual specification. The GeoUML modeling language contributes to the solution of these two issues by providing a set of predefined OCL templates for expressing the most common spatial constraints. In this paper, we deal with the validation problem by proposing a methodology for automatically translating the OCL constraints of GeoUML into SQL spatial queries, thus obtaining a platform independent general validation tool. The testing of the effectiveness of the already implemented OCL templates in ameliorating the quality of spatial data has been performed in the context of real life data production processes.

Categories and Subject Descriptors
H.2.8 [Database Management]: Database applications – Spatial databases and GIS.

General Terms
Management, Design, Verification.

Keywords
Conceptual modeling, spatial integrity constraints, spatial data validation.

1. INTRODUCTION
The validation of spatial integrity constraints specified at conceptual level is an important activity, both for checking the quality of the provided datasets resulting from a production process and for monitoring the consistency of information stored in a spatial database, in particular when updates have to be handled. Our experience in a project with the Lombardy Region (Italy) confirms the need for some kind of control procedures able to verify the quality of the provided data and their accordance with the conceptual specification. In addition, considering an open distributed environment, like the regional Spatial Data Infrastructure (SDI) of Lombardy, where the level of integration and interoperability is very high, there is a strong need for validation tools that guarantee the adherence of the data to a given common schema, also with respect to the spatial constraints. Finally we have observed that, since in this kind of distributed environment validation operations can be performed by applying different methods, it is necessary to ensure that any possible validation test has the same behavior.

The ability to define spatial integrity constraints at conceptual level allows designers to abstract from a particular implementation model and to apply one common constraint framework. However, the lack of a common data model for spatial data at logical level and the variety of data structures for spatial data needed by applications have created a consistent gap between the conceptual design of a spatial dataset and its implementation on GIS systems. A consequence of this situation is that the conceptual schema of a spatial database is a pure document with a weak binding to the data stored in the chosen GIS system. Moreover, the validation process of data properties becomes a very cumbersome problem, since conceptual properties are not mapped directly on the implementation structures and the chosen GIS system often introduces additional constraints on the spatial data representation that correctly are not specified at conceptual level.

In this paper we consider an ISO TC211 compliant approach that allows the definition of spatial integrity constraints into GeoUML conceptual models through the use of predefined OCL template [5]. Thanks to these templates the designer can specify topological and part-whole constraints in a straightforward manner, even if she does not know the OCL language details. The aim of this paper is to describe a validator tool that, starting from the formal specification of these constraints together with a XML representation of the conceptual schema, is able to verify the correctness of spatial datasets.

The paper is organized as follows: Section 2 discusses some previous results about the specification of integrity constraints.
with the OCL language and their translation into SQL queries. Section 3 reports our experience about the need for a validation approach. Section 4 highlights the rationale of our approach and the motivations for its differences with respect to the approaches analyzed in Section 2. Section 5 presents the general architecture of the tools involved in the proposed methodology. Section 6 deals with an example of the presented validation approach and Section 7 presents conclusions and future work.

2. RELATED WORK
In [1] Demuth et al. proposed a systematic study about the use of OCL invariants for the specification of database integrity constraints. The authors analyzed how OCL invariants on classes can be automatically translated into SQL assertions that search for objects which violate the constraints. For doing so they assumed a “classical” class-to-table mapping where each class is represented as a table, each many-to-many association is mapped into a distinct table, each one-to-many association is implemented with a foreign key in the table corresponding to the many side, while each one-to-one association is implemented by a foreign key in one of the two classes. The authors proposed a translation methodology based on the definition of eight different patterns, each one designed for encapsulating one different OCL language concept: invariants, basic types, classes and attributes, navigations, operations, complex predicates, basic values and collections. They concluded that only the OCL iterate construct cannot be mapped into declarative SQL. In [2] Demuth et al. extended their approach by using SQL views instead of SQL assertions during the translation of OCL queries, because assertions are not available in all DBMSs. The proposed translation method has been implemented into an OpenSource tool, called OCL2SQL tool [3] developed by the Dresden University of Technology. The core component of this tool is the SQL code generator which generates the SQL code for an OCL invariant based on the parsed, typechecked and normalized OCL expression given as an abstract syntax tree.

In [4] Dubois et al. dealt with the modeling and specification of topological integrity constraints in databases for checking the quality of spatial data and monitoring the consistency of information. They highlighted the need for a formal definition of spatial integrity constraints at conceptual level that have to be automatically translated into checking mechanisms inside databases. They considered the use of OCL as formal constraint language, in order to specify constraints independently from the platforms and reduce the gap between the conceptual and implementation levels. In particular, they proposed an extension of OCL with the 9-intersection model, called OCL9IM, and they specialized the OCL2SQL tool for translating constraints into SQL statements.

Since also the Standard ISO TC211 19109 “Rules for application schema” [10] recommends the use of the OCL language for specifying (spatial) integrity constraints at conceptual level, it seems that the current mainstream of solution for the spatial constraint problem is to define spatial integrity constraints in OCL at conceptual schema level and then automatically translate them into spatial SQL queries for validation. We will refer to this approach as the “generic spatial OCL” approach.

This paper follows to some extent this main stream; however, for a set of reasons which are explained in detail in Section 4, it takes a different approach on a fundamental aspect, which has several consequences. The approach presented here is called “GeoUML methodology”.

Before dealing with the motivations why the GeoUML methodology differs from the generic spatial OCL approach, we report in the next section about the experience which has been performed in order to determine the usefulness of the validation approach.

3. VALIDATION OF SPATIAL DATA: A CASE STUDY
Several experiences motivated the implementation of the tools of the GeoUML methodology. These experiences support the following assertions:

1. It is important to have a formal conceptual specification of spatial constraints in spatial data production.
2. It is important to have an automatic implementation of the validation procedures which check the conceptual constraints on the data.
3. The check of constraints is important for augmenting the quality of spatial data.

The most important of these experiences has been the project started in 2006 by Lombardy Region. This project aims at developing in a few years a completely new topographic Database of the whole region; the project has been subdivided into subprojects and each subproject is separately managed, but the same Data Specification is used by all subprojects and covers the following thematic layers: road and railway networks, bridges, dams, buildings, industrial sites, hydrographic network, administrative boundaries, vegetation, street names and numbers (in total about 114 feature types). Of course, the Data Specification has been slightly modified from time to time in order to ameliorate it, basing on the performed experience.

The first subproject, for example, was carried out during 2007 and concerned the production of geo-spatial datasets covering a portion of about 1065 Km², representing the 4.5% of the whole region. The Datasets (one for each feature type) were provided in shapefiles (ESRI open format). Globally, more than 1 million features were provided.

Regarding data validation the whole project has the following characteristics:

- Spatial integrity constraints were defined, but their specification was written in natural language and no formal model was used.
- Procedures were developed in order to check the constraints (in an ESRI ArcGis environment).
- A certain effort was done in order to emphasize in the official documents the importance of the constraints; the contractors knew that constraints would be checked by automatic procedures before acceptance; the contractors were supported in understanding the meaning of the specification via courses and on-line query answering.

This effort has been useful in elevating the data quality of the produced data; many errors have been discovered and corrected. Many errors were detected by the validation phase and in many cases data were rejected and contractors had to provide them
again. For example, considering the constraint requiring the
disjointness or adjacency among the geometries of the features of
a given feature type, in the first subproject 600 errors among
300000 features were found by an automatic procedure.

However the project has also shown the limitations of this
approach.

- The lack of a formal conceptual specification of
constraints has allowed different interpretations which
required a big effort for being disambiguated.
- Validation procedures had to be generated ‘ad hoc’ for
each specific constraint type. This effort was so high
that it was not possible to implement a validation
procedure for each informally specified constraint:
among about 200 constraints only about 100 constraints
have been checked, by trying to reuse the same
procedure for different feature types introducing
parameters. Even if ArcGIS provides the basic tools for
testing many spatial constraints, in some cases specific
procedures have to be implemented with an important
additional cost.
- Even small changes in the data specification, which
occurred for instance when passing from one subproject
to another one, caused a very high cost not only because
of the need to adapt the procedures, but also for the
difficulty of keeping track of the correspondence
between different versions of the specifications and of
the procedures.

From this experience we can observe that, without a direct
 correspondence between the conceptual definition and the
implementation of spatial constraints, they lose much of their
usefulness. Therefore, spatial constraints have to be transferred
into the implementation level through some techniques. The
easiest solution, applied also in the Lombardy project, consists in
implementing some ad-hoc procedures for a particular conceptual
schema and a particular GIS system. However, this solution
presents a main problem: it cannot be reused into another context,
for example with another schema or into another implementation
system. This becomes particularly critical in a SDI (distributed)
context, where the involved entities need that the same check
performed by different procedures in different places produces the
same results. Therefore, in this paper we propose a solution that
automatically produces validation procedures for conceptual
spatial constraints, independently from the particular system and
the particular logical/physical schema used for the
implementation.

4. RATIONALE OF THE GEOUML
METHODOLOGY

The fundamental difference between the GeoUML methodology
[5][6] and the generic spatial OCL approach reported in Section 2
is the following: instead of augmenting OCL with new spatial
operators, in GeoUML a set of spatially oriented OCL templates is
defined. These templates include not only the topological
relationships, but also the part-whole ones, i.e. strong and weak
composition, membership and partition, where composition and
membership are interpreted geometrically, not just structurally.
For instance, if we say that a Region is composedOf Provinces, we
mean that the geometric union of the surfaces representing the
Provinces must be equal to the surface which represents the
Region.

We have developed this approach in order to overcome a set of
limitations encountered in trying to apply the generic spatial OCL
approach. These difficulties are listed in Figure 1.

The difficulty 1.1 is important: our experience in several real life
projects suggests that domain experts have many difficulties to
specify and understand OCL constraints.

In [6] we have shown, in agreement with several other research
works, that part-whole constraints are important in expressing
spatial properties; hence we consider point 1.2 a limitation.
Similar considerations can be done regarding segmented attributes
and a few other abstractions which are needed in defining spatial
conceptual schemas.

<table>
<thead>
<tr>
<th>1. Limitations at conceptual level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1. OCL is difficult to use by users (who are typically</td>
</tr>
<tr>
<td>domain experts, not ICT technologists).</td>
</tr>
<tr>
<td>1.2. The integration of the 9IM with OCL, presented</td>
</tr>
<tr>
<td>in [4], does not cover part-whole constraints, which</td>
</tr>
<tr>
<td>are needed at conceptual level.</td>
</tr>
<tr>
<td>1.3. Abstractions like “segmented attributes”, which are</td>
</tr>
<tr>
<td>very useful in defining conceptual schemas, are not</td>
</tr>
<tr>
<td>treated by the general OCL approach.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2. Limitations at the implementation level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1. The OCL definition of some of the part-whole</td>
</tr>
<tr>
<td>constraints needs the use of the iterate operation,</td>
</tr>
<tr>
<td>which is not supported by the generic OCL approach</td>
</tr>
<tr>
<td>(see [1] [2]).</td>
</tr>
<tr>
<td>2.2. Some GeoUML constructs, such as the segmented</td>
</tr>
<tr>
<td>and subregion properties, need a physical mapping</td>
</tr>
<tr>
<td>that is more complex than the “classical” class-to-</td>
</tr>
<tr>
<td>table mapping, which is assumed by the generic</td>
</tr>
<tr>
<td>spatial OCL approach.</td>
</tr>
<tr>
<td>2.3. Generic OCL expressions with embedded spatial</td>
</tr>
<tr>
<td>functions and predicates are very difficult to optimize.</td>
</tr>
</tbody>
</table>

Figure 1. A list of limitations of the generic spatial OCL
approach.

In order to show how the GeoUML methodology helps in
overcoming the above difficulties, consider the following
example: there are two classes, called Region and Province; both
classes have a spatial attribute, of type “surface”, called territory.
In Figure 2 a spatial constraint between these two classes is
defined in natural language, in GeoUML and in OCL.

Notice that the OCL constraint has been generated by applying a
template to the formulation of the GeoUML constraint; in fact,
GeoUML constraints are just short forms for OCL templates.

This example shows that the GeoUML formulation is, at least in
cases like this one, much easier to understand than the
corresponding OCL formulation, so that it helps in overcoming
limitation 1.1; moreover, it shows an example of a part-whole
constraint which cannot be expressed in the approach described
in [4] (limitation 2.1) and, also, it is an example of the necessity of using the iterate constructs of OCL, which is not supported by the OCL2SQL approach defined in [2] (limitation 2.2).

The last point in Figure 1 (limitation 2.3) refers to the performance issue and becomes important if we have to check complex constraints on large spatial databases; as far as we know, there is currently no theory of generic spatial OCL constraints optimization and therefore it is very difficult to foresee whether a generic spatial OCL constraint will be evaluated in an acceptable time or not. Of course, it is much easier to optimize a set of constraint templates, because one knows a priori their structure, as we will show in Section 6.4.

4. At the implementation level the current GeoUML constraints are not strictly a subset of the generic spatial OCL approach, because part-whole constraints are implemented by the former and not by the latter (although they can be expressed at conceptual level).

5. STRUCTURE AND TOOLS OF THE GEOUML METHODOLOGY

The general architecture of tools involved in the GeoUML methodology is shown in Figure 3 and explained below. A tool called GeoUML Catalogue has been developed for storing a conceptual specification of a Spatial database consisting of a GeoUML conceptual schema, including constraints and supporting documentation. Since this paper is focused on data validation, we omit the description of the “documentation oriented” features of the GeoUML Catalogue.

Since GeoUML is used in order to define the constraints of a spatial database at conceptual level, and we want to control these constraints on real data, which is stored in a physical structure, we need to take care of the conceptual-to-physical mapping in order to apply the validation. There exists a large variety of logical and physical structures that can be produced by the mapping rules applied to a GeoUML schema. At a first level this variety is determined by the general implementation technology; some of the main technologies which are being considered in the current development of the GeoUML Catalogue are:

1. Transfer formats:
   - GML based
   - Shapefile based

2. Geo-relational Spatial Databases:
   - Oracle
   - PostGis

GML is a standard XML based language for representing geographical data, which was initially defined by the Open GeoSpatial Consortium [7] and then adopted also by ISO [8]. Therefore, it is the natural candidate technology for sharing and transferring geographical information independently from the particular GIS system chosen by an organization. Shapefiles are an ESRI open specification which still constitutes the most used transfer format for spatial files. On the other hand, Oracle Spatial and PostGIS are two of the most widely used spatial database systems in the commercial and open source areas.

At a second level, for each general implementation technology it is possible to define a different set of rules for the conceptual-to-physical mapping from a GeoUML schema to a Schema in the chosen technology. A given set of rules is called an Implementation Models (IM). A different Implementation Model is needed for each different general implementation technology; some flexibility can be provided in an implementation model for dealing with variations in the second level rules. Major variations, like the use of a topological structure or not, require the definition of different Implementation Models even for the same Database system.

In order to deal with this variety of possible Implementation Models, the GeoUML Catalogue is designed for being extendible with modules (plugin) which are specialized for automatically translating a GeoUML conceptual schema into a physical schema.
by applying one Implementation Model. These modules are collectively indicated as “Translator” in Figure 3. Currently, an Implementation Model for GML (by applying the encoding rules of the Standard ISO 19136 [11]), for Oracle Spatial and for PostGIS have been defined and implemented.

The data validation process is performed by a tool called GeoUML Validator. This tool uses the information managed by the Catalogue in order to read and validate datasets with respect to the conceptual specification. In order to deal with the variety of data types and formats, the GeoUML Validator integrates different module for different data formats. The general architecture of tools involved in the GeoUML methodology is shown in Figure 3.

Figure 3. The general architecture of tools involved in the GeoUML methodology.
Implementation Models, the validator is divided into two macro-modules:

(i) a set of loading modules, called GeoUML Loaders, one for each Implementation Model, that is able to load into a normalized geo-database, called internal representation (currently implemented on PostGIS), a dataset created according to the implementation model IM and

(ii) a constraints validator module, called GeoUML Tester, that converts the GeoUML constraints defined in the conceptual schema into SQL queries on the normalized geo-database and executes them, producing reports about detected violations.

In this way the variety of input structures are handled by the GeoUML Loader, which also checks the single geometry values with respect to the geometric domains declared in the GeoUML schema and takes into account the different representations used in the different GIS technologies (GML, Oracle, PostGIS, but also Shape File). The GeoUML Tester generates SQL queries from the OCL constraints specified in the GeoUML schema and executes them on the data loaded in the normalized geo-database; in this way the constraints checking is always performed using the same GIS technology and is independent from the format of the input datasets.

The choice of a geo-relational implementation for the internal representation of a GML dataset allows one to define the constraint checking procedures as a set of automatically generated spatial queries.

In the following section the general approach for generating these spatial queries from the conceptual integrity constraints defined in a GeoUML schema is presented on hand of a specific example. The example covers only one of the possible SQL templates that has been implemented in the GeoUML Tester, but it is a sufficiently complex example to illustrate the main aspects of the approach.

6. AN EXAMPLE OF TOPOLOGICAL CONSTRAINT CHECKING

This section considers a spatial integrity constraint between the classes Road and Bridge depicted in Figure 4. In particular, the constraint requires that each portion of Road with seat "On bridge" is contained in the union of the geometric extents of the Bridge class instances.

The problems faced in this example cover all the areas of complexity which has been faced in query generation: complexity of the conceptual-to-physical mapping, complexity in the constraint itself, and optimization. First, the seat of a road is represented as a segmented property, whose implementation into a geo-relational database is more complex than the "classical" class-to-table mapping. Moreover, the topological constraint is defined on the union of the geometries of the constraining class, therefore its OCL formulation requires the iterate construct. Finally, an optimization performed during the constraint translation is presented, which allows to apply the check also on large datasets.

Section 6.1 starts with the conceptual design of the previously mentioned topological constraint, using the GeoUML modeling language. In Section 6.2 the designed features are translated into database tables. Section 6.3 presents the necessary steps for translating the OCL constraint into a SQL query. Finally, Section 6.4 deals with the optimization issue.

6.1 GeoUML Conceptual Design

In this example we consider two spatial features: one denoting the bridge areas and one denoting the roads. In GeoUML these features are represented respectively by the two classes Bridge and Road of Figure 4. Note that the seat property of a road is represented through a segmented property, since the seat value changes along the road path.

A segmented property $A$ with domain $D_s$ is a property defined on the linear geometric attribute $g$ of a spatial feature $f$ whose value changes along $g$. Therefore, the value of a segmented property is not uniform for a feature, but it is a function of the spatial attribute $g$ of this feature. A segmented property subdivides the curve $g$ into segments of uniform value, called homogeneous segments.

In the graphical GeoUML notation, the road seat segmented property is represented through two methods:

$$\text{segmentsOf\_RoadSeat}(\text{condition on SEAT\_TYPE}) : \text{GU\_CXCurve}\times D$$

The first method returns a complex curve representing the set of curve segments that satisfy a particular condition on the attribute domain SEAT\_TYPE; while, the second method returns the value of the segmented property in a particular point.

In the textual GeoUML notation, segmented properties are specified placing the keyword segmented properties after the definition of the interested spatial attribute, followed by the list of segmented properties for this spatial component.

Once defined the two GeoUML classes, we can build the constraint between the Extent attribute of the Bridge class and the homogeneous segments of the Road class induced by the roadSeat segmented property. This topological IN constraint is
of type union, since it requires that the geometry of each homogeneous segment is contained into the union of the extents of all the Bridge instances. This constraint is specified using the GeoUML constraint templates as follows:

```plaintext
constraint
  Road.segmentsOf_RoadSeat(roadSeat='01') (IN)
  union Bridge.extent
```

and corresponds to the following OCL expression, obtained by applying the OCL template for the topological constraint union to the above specification:

```plaintext
context Road
inv:
  self.segmentsOf_RoadSeat(roadSeat='01') ->
  forall(t:GU_Object |
    t.check(In,
    Bridge.allInstances-> select(a:Y | a.roadSeat='01').f->
    iterate(b:GU_Object, 
    acc: GU_Object = NULL |
    acc.gUnion(b)))))
```

where the function: a.check({r1,...,rn},b)=def a.r1(b) or ...
or a.rn(b) checks if the disjunction of topological relations r1,...,rn is valid between the objects a and b. In GeoUML each spatial type implements a set of methods for checking the topological relations disjoint (DJ), touch (TC), in (IN), contain (CT), equal (EQ) and overlap (OV): which refine those defined by Clementini et al in [9]. GeoUML includes also the relations intersect (IT) and cross (CR), where cross can only be used with lines. Moreover, the function gUnion has been added to all classes representing the geometric types of GeoUML: it produces the pointset union of two geometries. The set of geometric types of GeoUML is called GeoUML Geometry Model and GU_Object is its root class.

As we can see this constraint uses the iterate operator, therefore it cannot be translated in SQL using the tool proposed in [1][2].

### 6.2 GeoUML Conceptual-to-Physical Mapping

Each GeoUML class is translated into a table, whose name corresponds to the class name and with an attribute for each property of the class, as illustrated below:

```plaintext
Table Road
  uuid: varchar(64)
  code: integer
  name: varchar(128)
  path: multilinestring
```

```plaintext
Table Bridge
  uuid: varchar(64)
  id: integer
  extent: multipolygon
```

GeoUML states that a segmented property like roadSeat, defined on the linear geometric attribute path of the class Road, can be physically represented in two ways: using a geometric representation of segments or using linear referencing. This level of abstraction of the conceptual notion of segmented property from the two main implementation approaches has proved to be important in SDIs; for instance, if one has to integrate different Road Networks which represent the same segmented attribute using the two different implementations, the common conceptual model can be defined in GeoUML and the difference in implementation is relegated in the corresponding Implementation Models.

However, since the representation based on linear referencing is not yet implemented in the GeoUML Validator, we refer here to the geometric representation of segments, which is implemented with a separate table containing the identifier of the corresponding Road instance, the value of the segmented property and the geometry of the homogeneous segments, that is the set of points where the segmented property roadSeat presents the stored value. In the illustrated case the following table will be produced:

```plaintext
Table Road_Path_SG
  uuid: varchar(64)
  id_Road: varchar(64)
  roadSeat: integer
  geometry: linestring
```

### 6.3 Constraint Mapping

For the translation of the specified OCL constraint, we first create two temporary tables, named Constrained and Constraining, which contain the sets of objects obtained by evaluating the constrained and the constraining expression respectively.

```plaintext
CREATE TABLE Constrained AS
  SELECT x.uuid, x.code, t1.roadSeat, ST_UNION(t1.geometry)as g
  FROM Road as x JOIN Road_Path_SG as t1
  On x.uuid = t1.uuid_Road
  WHERE g IS NOT NULL AND t1.roadSeat = '01'
  GROUP BY t1.roadSeat, x.uuid, x.code;

CREATE INDEX Constrained_G_Index ON
  Constrained USING GIST(g)

CREATE TABLE Constraining AS
  SELECT y.uuid, y.extent as f
  FROM Constraining y
create index Constrained_G_Index ON
  Constraining USING GIST(f)
```

The Constrained table contains the segments induced by the segmented property of the Road class where the roadSeat property has value ‘01’, that is ‘OnBridge’.

The union existential topological constraint refers to the union of the spatial attributes of one or more instances of the constraining class Bridge. Therefore, in order to retrieve the set of violating objects we have to perform the union of the geometries of the constraining class instances.

The SQL query that retrieves the violating objects becomes:

```plaintext
SELECT x.uuid as 'failedElements', x.g as 'failedGeometry'
FROM Constrained x
WHERE NOT ST_Within(u.g, ST_UNION(y.f)
  FROM Constraining y)
```

### 6.4 Query Optimization

Performing the union of a large number of geometries is an inefficient operation: some preliminary experiments using an increasing number of geometries with the ST_Union() function of PostGIS have been performed, showing that over the threshold of 5000 geometries the execution time is greater than 1 hour. In order to overcome this limitation we can observe that, to check the violation of a union existential topological constraint between
the spatial attribute $g$ of the constrained class instances and the union of spatial attribute $f$ of the constraining class instances, it is enough to check the violation with respect to the union of all the spatial attributes $f$ that have a nonempty intersection with $g$. Therefore, the geometric union has to be performed only on the necessary geometries, i.e. those that intersect the constrained geometry.

<table>
<thead>
<tr>
<th>Number of elements</th>
<th>Duration of the non optimized query</th>
<th>Duration of the optimized query</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>796 ms</td>
<td>531 ms</td>
</tr>
<tr>
<td>100</td>
<td>14,23 sec</td>
<td>4,98 sec</td>
</tr>
<tr>
<td>200</td>
<td>35,09 sec</td>
<td>10,05 sec</td>
</tr>
<tr>
<td>300</td>
<td>57,58 sec</td>
<td>15,14 sec</td>
</tr>
<tr>
<td>400</td>
<td>1,31 min</td>
<td>20,06 sec</td>
</tr>
<tr>
<td>500</td>
<td>1,67 min</td>
<td>24,69 sec</td>
</tr>
<tr>
<td>600</td>
<td>2,17 min</td>
<td>29,37 sec</td>
</tr>
<tr>
<td>700</td>
<td>2,56 min</td>
<td>32,47 sec</td>
</tr>
<tr>
<td>800</td>
<td>2,98 min</td>
<td>39,09 sec</td>
</tr>
<tr>
<td>900</td>
<td>3,56 min</td>
<td>44,00 sec</td>
</tr>
<tr>
<td>1000</td>
<td>4,06 min</td>
<td>47,55 sec</td>
</tr>
<tr>
<td>1500</td>
<td>6,76 min</td>
<td>1,23 min</td>
</tr>
<tr>
<td>2000</td>
<td>12,25 min</td>
<td>1,64 min</td>
</tr>
<tr>
<td>2500</td>
<td>17,03 min</td>
<td>1,95 min</td>
</tr>
<tr>
<td>3000</td>
<td>22,89 min</td>
<td>2,55 min</td>
</tr>
<tr>
<td>3500</td>
<td>31,52 min</td>
<td>2,94 min</td>
</tr>
<tr>
<td>4000</td>
<td>42,03 min</td>
<td>3,34 min</td>
</tr>
<tr>
<td>4500</td>
<td>54,78 min</td>
<td>3,73 min</td>
</tr>
<tr>
<td>5000</td>
<td>1,13 hour</td>
<td>4,16 min</td>
</tr>
<tr>
<td>5500</td>
<td>1,39 hour</td>
<td>4,64 min</td>
</tr>
<tr>
<td>6000</td>
<td>1,58 hour</td>
<td>5,00 min</td>
</tr>
</tbody>
</table>

For applying this optimization, we build a temporary table, named Constraining_Union, starting from the Constrained and Constraining ones, which contains for each instance $x$ of the constrained class, the spatial union of the instances of the constraining class $Y$ whose spatial attribute intersects the spatial attribute of $x$.

```sql
CREATE TEMP TABLE Constraining_Union AS
(SELECT x.uuid, x.g, ST_UNION(y.f), as f_union
FROM Constrained x LEFT JOIN Constraining y
ON ST_Intersects(x.g, y.f)
WHERE GROUP_BY x.uuid, x.g)
CREATE INDEX Constraining_Union_Index ON Constraining_Union USING GIST(f)
```

The SQL query that retrieves the violating objects becomes:

```sql
SELECT u.uuid as 'failedElements', ...
  u.g as 'failedGeometry'
FROM Constraining_Union u
WHERE NOT ST_Within(u.g, u.f_union);
```

Notice that this kind of optimization is at the same time very effective and very easy to determine thanks to the “predefined OCL template approach”, while optimizing generic OCL constraints is very difficult.

Table 1 shows the comparison between the duration of the non optimized query presented in the previous section and the duration of the optimized query presented above. Notice that, not only the duration of the optimized query is considerably smaller than that of the non optimized one, but also the duration of the optimized query increases less quickly with respect to the number of geometries (in fact, the behavior of the optimized query is almost linear).

7. CONCLUSION AND FUTURE WORK

In this paper we have proposed a methodology for automatically translating spatial integrity constraints expressed at conceptual schema level into SQL spatial queries. This methodology has been successfully implemented inside a platform independent validation tool that checks the conformity of a dataset with the conceptual specification.

We have confirmed the need for some procedures that automatically validate spatial integrity constraints defined at conceptual level through the experience of spatial data validation in a real project of Lombardy region in Italy.

We have highlighted some limitations of using pure OCL for specifying spatial integrity constraints both at conceptual and implementation level that justify the adoption of a different approach based on predefined constraint templates.

Finally, we have presented the architecture of the developed validation tool and a detailed example that starting from the conceptual design of a spatial integrity constraint, explains how it is translated into a SQL query.

For now the validation tool loads datasets stored into PostGIS and Oracle Spatial tables, but as a future work we plan to extend the loading function for accepting data in many other formats, such as GML or ESRI Shapefiles. Moreover, the validation procedure works currently on 2D geometries, but the extension to 3D geometries, limited to points and curves, is under development.

8. REFERENCES

Conference on Conceptual Modeling (ER), Austria (2005), pp. 465-482.


