ABSTRACT: In the last decade there has been an increasing interest in the use of tools for the creation of geographic maps. However, these tools include functionalities that allow users to insert information without verifying their consistency with the constraints defined by the designer. In this paper we propose an approach based on visual language parsing for guaranteeing the integrity of data produced during the map editing process. According to our approach, constraints are expressed by using high-level data model, such as OMT-G, and converted in a proper visual language grammar automatically. From such a grammar, a parser able to identify incorrect aggregation of spatial data input is generated. The grammar formalism also allows us to associate suitable semantic actions with productions in order to perform automatic corrections of inconsistent input data, error recognition and visualization, prompting actions for error recovery. We also present a system prototype supporting the proposed process.

Categories and Subject Descriptors
D.3.4 [Programming Languages]: Processors – parsing; translator writing systems and compiler generators; F.4.2 [Mathematical Logic and Formal Languages]: Grammars and Other Rewriting Systems – grammar types, parsing; H.2.8 [Database Management]: Database Applications – spatial databases and GIS.

General Terms: Geographical maps, spatial data, Visual lanaguage parsing

Keywords: Spatial Consistency, Integrity Constraint Checking, Data Quality, Visual Language Parsing, Grammar Formalism, Topological Relationships.

Received 9 June 2007; Reviewed and accepted 7 Sep. 2007

1. Introduction

In any geographical information system (GIS) the reliability of any result of queries, analysis or reasoning, depends on dataset quality. Current datasets contain largely inconsistent data also because commercial GIS do not assist users during the map editing process with adequate means for constraining input spatial data. On the contrary, they permit the insertion of inconsistent data through import function or simply “digitalization” operations. Later on, a range of correction functions may be applied to “clean up” the data, verifying its consistency. However, this approach does not create relationships between the process of data editing and the quality of obtained datasets. Consequently, users of such spatial datasets have no guarantee about the accuracy of the data they contain, even though they base their subsequent analysis on the assumption that data is error free or errors are kept to an “acceptable” level [8].

The quality of spatial data can be improved by imposing integrity constraints. As for spatial databases, in addition to traditional integrity constraints, rules about spatial data have to ensure the consistent updating of spatial data (i.e., consistency of the geometric representation of objects with respect to a spatial model). A typical classification of these spatial constraints is: topological, semantic and user defined [28].

Based on a set of mathematical predicates, topological constraints involve five relationships. They are disjoint, touches, crosses, within, and overlaps. Three additional relationships have been added to help users, namely contains, intersects, and equals [23]. As an example, the city’s area must be entirely contained into the administrative city limits.

Semantic constraints concern with the study of the meaning and use of words and phrases. Therefore, in this context, semantic integrity constraints deal with the constraints due to the meaning of geographic features. Typical examples are “buildings cannot be crossed by streets” and “users must not enter a road running through any body of water in a class of water objects including rivers and lakes”.

Finally, user defined constraints are more esoteric in nature and not necessarily based on semantics, and allow database consistency to be maintained according to user defined constraints analogous to business rules in non-spatial DBMSs [28]. For example, for external or legal reasons it may be desirable to locate a nuclear power plant at a given distance from residential areas.

To guarantee the effective verification of map constraints according to the designer requirements, it is necessary to automate the checking process upon data entry. To this aim, in this paper we present a visual language parsing approach for constraint checking of input spatial data. The integrity of data produced during the map editing process is guaranteed by a constraint checker automatically generated from a visual language grammar. In order to reduce the efforts for defining the constraints to be checked a high-level data model is used to specify the user needs. The process of building map constraint checkers consists of: 1) defining specific spatial constraints with a visual notation, as an example with an OMT-G diagram [7], 2) transforming the constraints into a visual language grammar, as an example into an Extended Positional Grammars (XPG) [12], and 3) automatically generating the constraint checker using the defined grammar. We also present a system prototype which completely supports the proposed process.

Our approach exhibits several advantages. In particular, the use of a grammar formalism allows us to exploit the well-established theoretical background and techniques...
developed for string and visual languages in the setting of geographical information systems, such as the "compiler-compiler" technique widely adopted for the generation of programming environments. Moreover, several tasks can be easily performed on the constraint grammar such as customization and modifications as well as maintenance and debugging. Suitable semantic rules can be defined to perform automatic corrections of inconsistent input data, error recognition and visualization, prompting actions for error recovery.

The paper is organized as follows: Section 2 presents a survey of recent related work. Section 3 shows the process for specifying the spatial integrity constraints and creating the suited checker. In detail, Subsections 3.1 and 3.2 describe the data model OMT-G for defining map constraints and the XPG formalism together with the mapping rules for mapping the OMT-G constraints into grammar productions, respectively. In Subsection 3.3 we describe the parsing technique used to identify violated constraints. Section 4 describes the system prototype implementing the proposed approach. Conclusions and further research are discussed in Section 5.

2. Related Work

Even if the constraint checking problem of geographic maps has been analyzed in the last decade, it is still evident the inability of current GIS regarding the implementation and management of spatial integrity [24, 30]. In this section, we present this work by highlighting the most important features. A constraint solver based on programming logic has been presented in [2]. The authors define a constraint system for handling spatial data types (points, lines and polygons) and constraints on them (equalities and inequalities, memberships, metric, topological and structural constraints), and provide a suitable theory for this constraint system. In [3] the same authors have presented a constraint solver implemented with transformation rules. These rules handle a special kind of constraints used for consistency checking, enabling an optimized and efficient resolution of spatial constraints.

In [18] Liu addresses the problem of consistency checking for Euclidean spatial constraints. In particular, a dimension graph representation is proposed to maintain the Euclidean spatial constraints among objects. The constraint consistency checking problem is transformed into a graph cycle detection problem on dimension graph.

A graph grammar formalism, which integrates both the spatial and structural specification mechanisms in a single framework is presented in [16] and [17]. In addition to nodes and edges, this formalism treats spatial constraints as a type of language constructs in the abstract syntax.

A methodology for spatial consistency improvement of geographical data sets in vector format has been described in [28]. It designs the process to discover inconsistency in three steps: error definition, error checking and error correction. The authors describe the first and the third one while the second one is disregarded. The properties are checked by means of computational geometry algorithms and the constraints are checked by first order calculus predicates.

In [20] a system developed for automatically maintaining topological constraints in a geographic database is presented. This system is based on extending to spatial data the notion of standard integrity maintenance through active databases. Topological relationships, defined by the user, are transformed into spatial integrity constraints, which are stored in the database as production rules. A similar approach is also used in [5].

In [26] Rigaux et al. present Dedale, a constraint-based spatial database system relying on a linear constraints logical model. This system provides both an abstract data model and a user declarative query language based on SQL in order to represent and manipulate geometric data in arbitrary dimension. A different approach which combines relational and constraint data models is used in [15]. In particular, Goldin presents a three-tier constraint database architecture which increases the level of abstraction between the physical data and its semantic, by introducing an additional layer to the classical relational data model architecture (logical and physical layer) in order to manage both constraint-based and geometric data representations in the same layer of abstraction, in opposition to the pure constraint databases, where all data are represented in terms of constraints.

Attribute grammars have been used in [29] for the recognition of patterns in geographic maps. In particular, the authors designed and implemented a system for defining abstract regions by hierarchical descriptions. The hierarchies are represented by attributed grammars that can be translated by a compiler of compiler to yield a parser for abstract regions. The abstract region candidates that were identified by the parsing rules can be evaluated to check if they conform to the definition provided by the user.

In the following of this paper, we will also refer to the spatial constraint definition of the editing phase. In particular, we will refer to the OMT-G modeling language. It is an object-oriented data model for geographic applications which provides appropriate primitives for representing spatial data, supports spatial relationships and allows the specification of spatial integrity rules through its spatial primitives and spatial relationship constructs. We have chose OMT-G for its capability of explicitly specifying the constraints in associations and attributes [14], which is a limitation of the models extending UML [27], such as Ext. UML [25] and GeoFrame [13]. Moreover, OMT-G seems to be the most simply and user friendly for non-expert constraint designers.

A limitation common to model-based approaches is their inability to specify certain properties. To overcome this problem, they have been combined with formal specification languages, such as OCL [22], providing the capability to express constraints in a textual form. These languages do not include mechanisms to enable model analysis and translation. This aspect is particularly important to support automatic correction of inconsistent input data, error visualization, and prompting actions for error recovery. On the other hand, grammar-based approaches are more general, and often rely on rigorous syntax and semantics, which provide powerful mechanisms to fully specify visual notations and their properties, including translation rules. Moreover, the difficulties in the process grammar specification often are overcome through automated support tools.

In the recent years, many different grammar-based approaches have been introduced for specifying and parsing visual languages [19]. The strong point of XPGs with respect to the other approaches is its capability to provide an efficient parser without limiting the modeling power [12].

3. The Map Constraint Verification Approach

In this section we describe the proposed technique for verifying integrity constraints during map editing. It allows us to exploit the well-established theoretical background and techniques developed for string and visual languages in the
setting of geographical information systems. In particular, grammars are used for specifying the user constraints and for generating the constraint checkers automatically. The editors invoking these checkers provide freehand editing as well as syntax-directed editing. The former allows users to draw incomplete and incorrect maps, and to insert spatial objects without any fixed order, postponing the validation phase at user will. The latter is supported by invoking the checker at each editing step, which rejects the editing actions violating integrity constraints. In the following we describe how to construct checkers supporting freehand editing, while in Section 3.4 we discuss how to integrate syntax-directed editing capabilities into checkers.

The process for the definition of map constraints and their verification is organized in three phases (see Fig. 1):
1) the constraint specification phase, where designers define the constraints on the spatial data of the edited maps;
2) the constraint checker generation phase, where the parser of spatial objects is generated starting from the specified constraints;
3) the constraint verification phase, where the user data input is verified by the constraint checker.

The construction process of a constraint checker starts by defining the constraints to be verified in terms of visual grammar productions. This process can be simplified by constructing editors that allow designers to both visually specify spatial objects and constraints among them, and automatically generate the corresponding grammar productions.

Fig. 1. The specification and generation process of a parser for checking map constraints during editing sessions.

In order to highlight the effectiveness of our process, in the following we describe:
1. how to specify constraints by means of OMT-G diagrams;
2. how to obtain the visual productions corresponding to an OMT-G diagram;
3. how to identify constraint violations through the generated constraint checker.

3.1 The OMT-G Data Model

Fig. 2 shows an OMT-G diagram modeling a typical real-world scenario of municipality field. In this example, several topological constraints are highlighted along with the geo-object classes. As an instance, the Municipal boundaries are a spatial aggregation of Health district, as well as Blocks are an aggregation of Parcels. The Health districts contain both Families and Blocks, while Built Parcels contain Addresses. As for the semantic constraints and the user defined constraints, they are not explicitly described by the OMT-G example, but they may be easily inferred by the common life. Semantic constraints can be: the Street segments cannot cross the Blocks and the Addresses must be contained in the Blocks, while a user defined constraint can be: an Unoccupied parcel must not contain any Address.

As for the semantic constraints and the user-defined constraints, they are not explicitly described by the OMT-G example, but they may be easily inferred by the common life. Semantic constraints can be: the Street segments cannot cross the Blocks and the Addresses must be contained in the Blocks, while a user defined constraint can be: an Unoccupied parcel must not contain any Address.
3.2 Specifying Map Constraints with Visual Grammars

Let us now describe the grammar formalism that we will used to describe the constraints between spatial objects. Extended Positional Grammars (XPG, for short) are a formalism developed in the visual language field and represent a direct extension of context-free string grammars, where more general relations other than concatenation are allowed [12].

The idea behind the definition of such formalism has been to overcome the inefficiency of visual language parsing algorithms by searching suitable extensions of the well-known LR technique. Moreover, the formalism allows modeling some critical visual languages, since it provides mechanisms to cope with context and introduce particular conflict-handling techniques within the parser.

In order to represent visual sentences, the XPG formalism uses an attribute-based approach [9]. In this approach a sentence is conceived as a set of attributed symbols. The values of the syntactic attributes are determined by the relationships holding among the symbols. Thus, a sentence is specified by combining symbols with relations. XPG productions have the following format:

\[ A \rightarrow x_1 R_1 x_2 R_2 \ldots x_m R_m \Delta, \Gamma \]

where \( A \) is a non-terminal symbol, each \( x_i \) is a terminal or nonterminal symbol and each \( R_i \) is a sequence of relations partitioned in two sub-sequences

\[ ([\text{REL}_1^1, \ldots, \text{REL}_k^1], [\text{REL}_1^n, \ldots, \text{REL}_k^n]) \]

The relation identifiers in the first sub-sequence of an \( R_i \) are called driver relations, whereas the ones in the second sub-sequence are called tester relations. Driver relations are used during syntax analysis to determine the next symbol to be scanned, whereas tester relations are used to check whether the last scanned symbol (terminal or non-terminal) is properly related to previously scanned symbols. Without loss of generality we assume that there are no useless symbols, and no unit and empty productions [1].

\( \Delta \) is a set of rules used to synthesize the values of the syntactic attributes of \( A \) from those of \( x_1, x_2, \ldots, x_m \).

\( \Gamma \) is a set of triples \( \{(T_p, Cond, \Delta)\}_{p=1}^{t} \), 0, used to dynamically insert new terminal symbols in the input visual sentence during the parsing process. In particular, \( T_p \) is a terminal symbol to be inserted in the input visual sentence; \( Cond \) is a pre-condition to be verified in order to insert \( T_p \); \( \Delta \) is the rule used to compute the values of the syntactic attributes of \( T_p \) from those of \( x_1, \ldots, x_m \).

As a consequence, if for each production \( A \rightarrow x_1 \ldots x_m \Delta, \Gamma \) of a grammar \( G \) the number of triples in \( G \) whose conditions can simultaneously evaluate to true is less than \( m-1 \), then the convergence of the parsing algorithm based on \( G \) is guaranteed.

In case of geographic maps, we have that the terminal symbols of the grammar are the spatial objects and the constraints are defined by the productions of the grammar. Moreover, the relations between the spatial objects can be expressed on the syntactic attribute, named \( bound \), which corresponds the boundaries of the objects. As an example, let us consider the constraints specified in the OMT-G diagram of Fig. 2. The following XPG grammar productions describe the topological constraints specified between the geo-object classes: Municipal boundaries, Health district, Family, Visit site and Route segment.

![Figure 3. A municipal map with the corresponding legend.](image)
The following productions specify the semantic constraint: the Addresses must be contained in Blocks, and the user defined constraint: an Unoccupied parcel must not contain any Address.

(10) HealthDistrict → DISTRICT <contains> Block
Δ:: HealthDistrict_bound = DISTRICT_bound
(11) Block → BLOCK <contains> Parcels
Δ:: Block_bound = BLOCK_bound
(12) Block → BLOCK
Δ:: Block_bound = BLOCK_bound
(13) Parcels → Parcel <sibling> Parcels
Δ:: Parcels_pos_set = Parcels_pos_set + Parcel_pos
(14) Parcels → Parcel
Δ:: Parcels_pos_set = Parcel_pos
(15) Parcel → UNOCC-PARCEL
Δ:: Parcel_pos = UNOCC-PARCEL_pos
(16) Parcel → BUILT-PARCEL <contains> Address
Δ:: Parcel_pos = BUILT-PARCEL_pos
(17) Parcel → BUILT-PARCEL
Δ:: Parcel_pos = BUILT-PARCEL_pos

The grammar productions describing the topological constraints of an OMT-G diagram can be automatically obtained by applying suitable mapping rules. As an example, if the diagram includes a 1-to-1 directional relationship contains between two geo-object classes a and b then the production “A → a <contains> b” is added to the grammar, where is A is a non-terminal created from rules applied to geo-object class a. In case of the relationship is 1-to-n then the productions

- A → a <contains> B,
- B → b <sibling> b, and
- B → b

are added to the grammar, where B is a new non-terminal. Other mapping rules are described in the Appendix.

A set of semantic routines can be associated to the XPG productions. Such routines are small pieces of code that accomplish semantic checks and translation tasks, and are each executed when the parser reduces the associated XPG production. As an example, in [12] we added semantic rules to XPG productions to automatically translate statechart diagrams into the XML Model Interchange (XMI) format [22], a standard file format for saving and loading UML designs. Additionally, each semantic routine can contribute to the construction of a syntax structure summarizing the information of the input spatial objects, which can be used to statically verify particular properties. As an example, we can identify anomalous behaviors of reactive systems modeled through statecharts by verifying the presence of loop transitions, conflicting transitions, and so on [11].

In the GIS domain, we can exploit this ability in order to verify some particular problems connected with the semantic of maps. As an instance, let us suppose we are using the “spaghetti” structure [23] for representing a road network and an alphanumeric attribute to store street directions. A semantic constraint to be checked is whether for each crossroad there exist at least an outgoing road and an ingoing road. Fig. 4(a) depicts an example of road network which violates this semantic constraint.

In order to accomplish such static verification, the syntax tree of Fig. 4(b) which logically summarize the information of the map in Fig. 4(a) can be easily produced by semantic rules (in brackets) associated to the following productions:

Cross → ROAD <Touches> ROAD
{ Cross.left = ROAD; Cross.right = ROAD;
  if (ROAD.dir == ROAD'.dir) then Cross.dir = ROAD.dir; else
  Cross.dir = “in/out”; }

Cross → Cross’ <Touches> ROAD
{ Cross.left = Cross’; Cross.right = ROAD;
  if (Cross’.dir == ROAD.dir) then Cross.dir = ROAD.dir; else
  Cross.dir = “in/out”; }

where:

1) the dir attribute of a ROAD node represents the direction of ROAD with respect to the crossroad, and
2) the dir attribute of a Cross node summarizes the directions of its ROAD leaves. In particular, the value in is assumed when all the dir node attributes are in, out if all the attributes are out, in/out otherwise .

The violation of the constraint is detected by checking whether the value of the dir attribute associated to the top Cross node is either in or out.

![Figure 4. A network road violating the constraint “at least an outgoing road has to exist in a crossroad” (a) and the corresponding syntax graph (b).](image)

Finally, semantic routines can also be used to provide a dynamic behavior to the input sentences, such as, coloring, moving and resizing objects, and so on.

3.3 Constraint Checking through Visual Language Parsing

We now describe the XpLR parsing technique that is used in our approach to generate parsers for the XPG describing constrained maps [12]. It is a technique for implementing visual systems based upon XPGs and LR parsing [1]. An XpLR parser scans the input in a non-sequential way, driven by the relations used in the grammar.

The components of an XpLR parser are shown in Fig. 5 and are detailed in the following.

The input to the parser is a dictionary, named Dp, storing the attribute-based representation of a map as produced by the visual editor. No parsing order is defined on the spatial objects in the dictionary. The parser retrieves the objects in the dictionary through a find operation, driven by the relations in the grammar. The parser implicitly builds and parses a linear graph (b) for an explicit relation labeled node is either in

Finally, semantic routines can also be used to provide a dynamic behavior to the input sentences, such as, coloring, moving and resizing objects, and so on.

3.3 Constraint Checking through Visual Language Parsing

We now describe the XpLR parsing technique that is used in our approach to generate parsers for the XPG describing constrained maps [12]. It is a technique for implementing visual systems based upon XPGs and LR parsing [1]. An XpLR parser scans the input in a non-sequential way, driven by the relations used in the grammar.

The components of an XpLR parser are shown in Fig. 5 and are detailed in the following.

The input to the parser is a dictionary, named Dp, storing the attribute-based representation of a map as produced by the visual editor. No parsing order is defined on the spatial objects in the dictionary. The parser retrieves the objects in the dictionary through a find operation, driven by the relations in the grammar. The parser implicitly builds and parses a linear graph (b) for an explicit relation labeled node is either in

Finally, semantic routines can also be used to provide a dynamic behavior to the input sentences, such as, coloring, moving and resizing objects, and so on.

3.3 Constraint Checking through Visual Language Parsing

We now describe the XpLR parsing technique that is used in our approach to generate parsers for the XPG describing constrained maps [12]. It is a technique for implementing visual systems based upon XPGs and LR parsing [1]. An XpLR parser scans the input in a non-sequential way, driven by the relations used in the grammar.

The components of an XpLR parser are shown in Fig. 5 and are detailed in the following.

The input to the parser is a dictionary, named Dp, storing the attribute-based representation of a map as produced by the visual editor. No parsing order is defined on the spatial objects in the dictionary. The parser retrieves the objects in the dictionary through a find operation, driven by the relations in the grammar. The parser implicitly builds and parses a linear graph (b) for an explicit relation labeled node is either in

Finally, semantic routines can also be used to provide a dynamic behavior to the input sentences, such as, coloring, moving and resizing objects, and so on.

3.3 Constraint Checking through Visual Language Parsing

We now describe the XpLR parsing technique that is used in our approach to generate parsers for the XPG describing constrained maps [12]. It is a technique for implementing visual systems based upon XPGs and LR parsing [1]. An XpLR parser scans the input in a non-sequential way, driven by the relations used in the grammar.

The components of an XpLR parser are shown in Fig. 5 and are detailed in the following.

The input to the parser is a dictionary, named Dp, storing the attribute-based representation of a map as produced by the visual editor. No parsing order is defined on the spatial objects in the dictionary. The parser retrieves the objects in the dictionary through a find operation, driven by the relations in the grammar. The parser implicitly builds and parses a linear graph (b) for an explicit relation labeled node is either in

Finally, semantic routines can also be used to provide a dynamic behavior to the input sentences, such as, coloring, moving and resizing objects, and so on.

3.3 Constraint Checking through Visual Language Parsing

We now describe the XpLR parsing technique that is used in our approach to generate parsers for the XPG describing constrained maps [12]. It is a technique for implementing visual systems based upon XPGs and LR parsing [1]. An XpLR parser scans the input in a non-sequential way, driven by the relations used in the grammar.

The components of an XpLR parser are shown in Fig. 5 and are detailed in the following.

The input to the parser is a dictionary, named Dp, storing the attribute-based representation of a map as produced by the visual editor. No parsing order is defined on the spatial objects in the dictionary. The parser retrieves the objects in the dictionary through a find operation, driven by the relations in the grammar. The parser implicitly builds and parses a linear graph (b) for an explicit relation labeled node is either in

Finally, semantic routines can also be used to provide a dynamic behavior to the input sentences, such as, coloring, moving and resizing objects, and so on.
Each row is composed of an ordered set of one or more sub-rows each corresponding to a parser state. The action and goto sections are similar to the ones used in LR parsing tables for string languages [1], whereas the next section is used by the parser to select the ones used in LR parsing tables for string languages [1], whereas the next section is used by the parser to select the next symbol to be processed. An entry next[k] for a state s contains a pair (R_driver, x), which drives the parser in selecting the next symbol y that is reachable from x, by using the sequence of driver relations R_driver.

Fig. 6 (a-i) show the steps to reduce the map of Fig. 3 through the parser generated from the extended positional grammar shown in the previous section. In particular, dashed ovals indicate the handles to be reduced, and their labels indicate the productions to be used. The reduction process starts by applying production 7 to a family object. This causes the terminal FAMILY to be reduced to the non-terminal Family. Due to the D rule of production 7, Family inherits the boundaries of FAMILY. Fig. 6(b) shows the resulting visual sentential form, and highlights the handle for the application of production 9. The objects SITE, LINE, and SITE' are then reduced to the new non-terminal VisitSite. Due to the D rule of production 9, the new VisitSite has the same boundaries of SITE.

![Figure 5. The architecture of an XpLR parser.](image)

![Figure 6. The reduction process for a municipal map](image)
Moreover, due to the G rule, since $|\text{SITE}_{\text{bound}}| = 0$, no new node SITE is inserted in the input. Similarly, the application of production 6 replaces the non-terminal Family symbol of Fig. 6(b) with the non-terminal Families. Fig. 6(c) shows the resulting visual sentential form. After the application of productions 8, 5 and 7 the visual sentential form reduces to the one shown in Fig. 6(e). Then, production 5 reduces the non-terminals Families and Family to a new non-terminal Families. Fig. 6(c) shows the resulting visual sentential form. After the application of productions 8, 5 and 7 the visual sentential form reduces to the one shown in Fig. 6(e). Then, production 5 reduces the non-terminals Families and Family to a new non-terminal Families. By applying the D rule of production 5, the new Families inherits all the boundaries of the reduced non-terminals (see Fig. 6(f)). The subsequent application of productions 7, 6, 4, 5, 3, 4, 2 and 1 reduces the map to the one shown in Fig. 6(i), confirming that the municipal map satisfies all the constraints specified in the grammar.

In case the user inserts a new family over a health district boundary as shown in Fig. 7, the parser is not able to apply production 4 since the family symbol does not satisfy the contains relationship. Thus, the parser terminates the process with an input symbol unparsed and either generating an error or executing the defined action.

3.4 Checkers supporting Syntax-directed Editing

Once integrated into map editors, the checker typology described in the previous section do not force the user to edit spatial objects according to the defined constraints but it just checks the integrity of maps when required by the user. However, this may turn out to be a disadvantage since they do not offer explicit guidance to the user. On the other hand, syntax-directed editors help decrease the number of errors and the duration of editing by preventing errors or by informing the user of their mistakes as the spatial objects are being entered. This editing mode requires an internal model that is modified by a set of editing operations that modify the meaning of the map.

The syntax-directed editing approach can be implemented by using an incremental parser generated from an XPG using the approach proposed in [10]. However, we can support both editing modes at the same time by defining an editing constraint checker that inhibits editing commands (such as rules for modification, deletion, etc. of spatial objects) violating simple map constraints, similarly to what has been done in [4,21], and using a XpLR parser invoked at the user will.

The editing constraint checker can be supported through XPG grammars by defining the inhibited editing commands with XPG productions. As an example, to forbid users to insert an Address in an Unoccupied parcel the following production should be added to such a grammar:

```plaintext
Parcel UNOCC-PARCEL<contains> Address
{
  Message « An Address cannot be placed inside an Unoccupied parcel »
}
```

When the production is applied a message is visualized to the user and the Address symbol is moved to a “safe” place, i.e. a position that does not violate any constraint.

4. A System Prototype for Map Editing and Integrity Constraint Validation

To prove the effectiveness of our approach we have implemented a system prototype able to support the process described in the previous section according to the syntax-directed editing mode. As shown in Fig. 8 the prototype is composed of two editing modules: the Constraint Editor and the Map Editor, and a data management system. The former enables the generation of grammars by allowing users to compose the OMT-G diagrams into a working area with textual annotations describing user-defined constraints (Fig. 9 shows a screenshot of the editor). Since this module presents very specific characteristics, we developed it from scratch.

The data management system and the Map Editor have been implemented by taking advantage from some largely diffused open technologies, such as PostGIS/Postgres™ and QuantumGIS™, which have been extended to integrate them with the proposed constraint checking functionalities. PostGIS adds support for geographic objects to the PostgreSQL object-relational database. Indeed, PostGIS “spatially enables” the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems. Quantum GIS (QGIS) is a user friendly Open Source GIS that runs on several platforms and supports vector, raster, and many database formats. Actually, it may be easily connected with many spatial DBMS, such as PostGIS, Oracle, etc., and provides support to the spatial data editing. In our system, QGIS can be replaced by any kind of spatial editor which is able to handle geometries interacting with PostGIS.

![Figure 8. The architecture of the system prototype](image)

![Figure 9. The main window of the Constraint Editor](image)
The Constraint Editor allows the constraint designer to visually compose OMT-G diagrams by the palette of symbols on top of the main window, as shown in Fig. 9. The grammar automatically generated from the sketched OMT-G diagram is translated into a “C” code program, and compiled to produce an object module (named Map Parser). This module is integrated within the PostGIS DBMS. Actually, we have implemented Map Parsers supporting the syntax-directed editing mode of the maps only, i.e., they are invoked as the user update the Spatial DB.

The main component of Constraint Editor is the toolbar located on the top of the window. It is composed of several buttons, each one devoted to invoke some operations. Starting from the left side, the first group of buttons provide basic functionalities such as opening a new working area, saving and printing their contents. The second and third button groups help the user to compose the OMT-G diagrams. Indeed, they contain the entity and the relationship OMT-G symbols which allow to select and draw the respective symbols into the working area. The compiler button generates Map Parser modules and uploads them into the spatial DBMS. Finally, some functionalities have been implemented to export the OMT-G diagrams into XML, C or JPG.

The Map Parsers supporting the syntax-directed editing are activated by PostGIS at each editing step, i.e., it works like a trigger on the update statements. As a consequence, the parser will return a constraint error in case the new geometry violates the grammar, otherwise it will allow the spatial DB update.

Fig. 10 depicts a screenshot of the Map Editor. The Legend component on the left contains the layers that compose the map. On the right is visualized the map loaded by the user. The error message highlighted in Fig. 10 is resulting from an insertion which does not satisfy the same constraint violated in Fig. 7.

![Figure 10. The Map Editor with a constraint error message](image)

### 5. Conclusions and Future Work

This paper proposed visual language grammars as means for building parsers able to verify spatial integrity constraints during the map editing phase. The sequence of steps necessary to design constraints, generate the parser and integrate it into editing tools has been presented by means of a typical real world example of a municipality scenario. In particular, the scenario has been depicted by means of an OMT-G diagram, which is automatically translated into suitable XPG productions. We have also presented a system prototype implementing the proposed process.

Our approach presents several advantages such as the easy customization and modifications of the constraint system. Another important feature is the capability of specifying simultaneously constraints concerning with more than two spatial objects. This ability enables us to describe very complex real world concept not manageable through simple visual compositions.

The next step in this approach is to verify the opportunity to use the XPG grammar to describe spatial patterns and recognize their presence within already existing maps. This will allow us to automate some onerous spatial search problems such as eventual dangerous scenarios or construction abuses.

We also plan to perform thorough usability experiments. Initial experimental uses of the system have already shown a potential enhancement in the user support during the constraint specification process. We plan to carry out an empirical evaluation of the constraint checker and a usability study of the developed system focusing on the use of XPG by non specialist users.

Finally, we are investigating the possibility to visually specify the constraints by using the GeoUML data model [6], which has receiving more and more attention by the GIS community for its compliance with International standards.

### References


Appendix: Rules for the derivation of a grammar from an OMT-G diagram

In this appendix we show some significant rules (to be applied in the specified order) used to obtain an XPG grammar $SK = (N, T, P)$ from an OMT-G diagram that describes a constrained map, where $N$ will be the set of non-terminal symbols, $T$ the set of terminal symbols and $P$ the set of productions.

Rule 1

For each non-specialized or non-aggregated geo-object class $cName$ insert in $T$ the symbol $cName$ with attribute $bound$ corresponding to the bounding box of the class.

Rule 2

For each generalization with $Gname$ as the generalized geo-object class and $PNAME$ as the specialized class:
- insert symbol $Gname$ in $N$;
- associate the attribute $bound$ to class $Gname$;
- add $bound$ to $PNAME$;
- insert $Gname \sqsubset PNAME$ in $P$;
- if $PNAME$ is not in $T$ then
  o add $PNAME$ in $N$;

Rule 3

For all aggregation or composition associations between a whole geo-object class $wName$ and part classes $pName_1, ..., pName_n$,

$\begin{array}{c}
\text{wName} \\
\text{pName}_1 \\
\vdots \\
\text{pName}_n
\end{array}$

where $j_1, z_1, ..., j_n, z_n$ represent the multiplicities of the associations and $j_1, ..., j_n$ are greater than 0,
- insert $Wname$ in $N$ and associate to it the attribute $bound$ of class $wName$;
- insert $Wname \sqsubset pName_1, ..., pName_n$ in $P$;
- insert $Wname \sqsubseteq pName_1$ $\sqcap$ $pName_2$ $\sqcap$ ... $\sqcap$ $pName_n$ in $P$ where $\sqsubseteq$ is a relation that is always true among any two symbols;
- if $pName$ is not in $T$ then add $pName$ to $N$ and associate to its attributes of the class $pName$.

If the multiplicity at the beginning of the association (on the border of $wName$) is $0..1$ then insert $Wname \sqsubseteq e \in P$.

Informally, the multiplicity of the associations is used to determine the number of the grammar symbols in the right-hand side of the productions and the number of productions with a given grammar symbol in the left-hand side.

Vincenzo Del Fatto is a PhD student both at the Department of Mathematics and Informatics (DMI) University of Salerno (Italy) and at LIRIS Lab (Laboratoire d’Informatique en Images et Systèmes d’Information) of INSA (Institut National des Sciences Appliquées Lyon (France). His main interests include, but are not limited to Database Management Systems, Geographical Information Systems (GIS) and Web GIS.
Metadata, Spatial Data Mining and Knowledge Discovery in Databases, Decision Support System, Visual Languages.

Vincenzo Deufemia graduated in Computer Science (cum laude) in 1999. He received the PhD degree in Computer Science from the University of Salerno in 2003. Since March 2006 he is assistant professor in the Department of Mathematics and Informatics at Salerno University. His main research focuses on software engineering, sketch understanding, visual languages, parsing technologies, and multimedia databases. On these topics he published several peer-reviewed articles in international journals, books, conference and workshop proceedings. He has served as program committee member for several international conferences.

Luca Paolino is a fellow researcher at the Department of Mathematics and Informatics (DMI), University of Salerno (Italy). He received his PhD in Computer Science in 2004 from the University of Salerno. His main interests include, but are not limited to Database Management Systems, Geographical Information Systems (GIS), Spatial Database Architectures, Web GIS, Geographic Metadata, Decision Support System, Visual Languages, Human Computer Interaction, Spatial Data Retrieval, Image Processing.