This paper deals with today’s state of the art in spatial database modelling. In particular, it discusses visual languages and related research. Major trends are described and an innovative language, termed second generation, based on a new concept called Spatial PVL is presented. This Plug-in for Visual Languages is used to extend the object-oriented standard language UML, and is implemented in a new freeware CASE tool called Perceptory.

Introduction

The last twenty years have witnessed widespread development of spatial databases using GIS (geographic information systems) or CAD technologies (computer-aided design) coupled with RDBMS (relational database management systems). Recent evolution of technology has introduced several alternative methods for spatial database building, including universal servers with spatial modules and viewers, spatial engines or components accessed through an API (application programming interface), Web servers with map viewing capabilities, data warehouses, and multidimensional databases. Especially component-based system development is rapidly changing the way we implement spatial databases. The traditional two-tier client/server architectures are being replaced by more complex, distributed multi-tier architectures. This diversity of solutions, a stronger integration with mainstream information technologies, as well as the increasing complexity of system architectures, are driving spatial database developers towards the general adoption of visual database modelling.

Visual database modelling helps us to understand and to describe more precisely the intended content of a client’s database. It also helps us to master the complexity of the problem, to facilitate the exchange and the validation of ideas, to improve the programming process, and to ease the maintenance of the system. In other words, database models can be thought of as thinking tools, communication tools, development tools, and documentation tools.

Database models are called conceptual when they depict the client’s database purpose and physical when they describe its software implementation. Underlying strategies used in these multi-level E/R (Entity/Relationship) and OO (Object Oriented) database modelling processes are described by Batini et al. [1992]; Cook and Daniels [1994b]; Rumbaugh [1996]; Bédard [1999]. An example of the conceptual level database model is presented in Figure 1. Its schema and dictionary are based on the Oracle Entity/Relationship language, or formalism.

The objective of this paper is to present today’s trends in spatial database modelling and to introduce a second generation of visual languages and CASE tools. The paper starts with an overview of the present state of the art of spatial database modelling focussing on the 1990s and onwards, introducing the first generation of visual languages developed for spatial databases. Then follows a description of the major trends influencing the way we will build spatial database models in the near future. These trends are used to introduce an innovative concept called Spatial PVL (Plug-in for Visual Languages). The first Spatial PVL is presented. Finally, we show how a second-generation visual language for spatial databases can be integrated into the Unified Modeling Language (UML) and can be implemented into a new CASE tool called Perceptory.
The First Generation of Visual Languages for Spatial Databases and the New Trends

Several visual modelling languages exist, the most recent ones relying on the object-oriented (OO) paradigm. Popular examples include Coad and Yourdon [1991a, 1991b]; Rumbaugh et al. [1991]; Shlaer and Mellor [1991]; Jacobson et al. [1993]; Martin and Odell [1993]; Booch [1994b]; Cook and Daniels [1994]. The last three years have seen a unification of efforts to develop the Unified Modeling Language (UML) (see Booch et al. [1999] for a complete description). UML is a de facto and official standard recognized by the Object Management Group. The only contender to UML is the Open Modeling Language (OML) described by Firesmith et al. [1998]. However, OML doesn't benefit from the same momentum as UML does, and it looks like UML is taking industry and academia by storm.

UML is intended for any type of software development. It can be used for databases irrespective of whether the implementation is done with OO or other database technology (as with most GISs). In the latter case, OO models simply must be translated into database units such as tables, columns, and keys.

Research in recent years has made database modelling more efficient than ever before, with several researchers focusing their work on spatial databases. The most important achievements include the following: (see Claramunt et al. [1997] for a comparison of the OO solutions).

- CONGGO [Pantazis and Donnay 1996]
- Geo-ER [Hadzilacos and Tryfona 1997]
- Geo-OM [Tryfona et al. 1997]
- GeoOOA with its modelling software [Kösters et al. 1997]
- MADS with its modelling software [Parent et al. 1997]
These researchers have extended existing languages or developed new languages to overcome a number of problems of applying non-spatial modelling languages to spatial databases. The most important problems include inefficient modelling of the objects’ possible geometries, of their geometric evolution over time, of their spatial aggregation and generalization, and of their spatial integrity constraints. Additional concerns include the search for the efficient separation or merging of the users’ view and the programmer’s view, of the semantic view and the geometric view, of attribute data and geometric data, of the map objects and the map aesthetic artefacts, and of the geometric data and their metadata. In many cases, the search for spatial solutions comes not from the impossibility to use non-spatial languages, but from the desire to improve significantly the efficiency of non-spatial languages. For example, reductions of 25% to 50% in model size are frequent with good spatially-aware solutions, significantly reducing efforts required to build and edit a database model, while improving its readability.

Some solutions included visual modelling software, or CASE tools (Computer-Assisted Software Engineering). The fundamental concepts relating CASE tools to spatial databases have been described by Bédard and Larivée [1992; Bédard [1999]. The most popular CASE tools also have been compared with regard to their appropriateness for spatial databases by Pouliot et al. [1997]. Popular CASE tools usually offer limited extension capabilities, their graphical and reporting capabilities are minimal, and none of them provide automatic code generation for GIS or other spatial database technologies. This explains why researchers have found it necessary to develop their own spatial CASE tools.

From an historical and Canadian perspective, it is of interest to know that Modul-R and Orion, developed at Laval University Center for Research in Geomatics, were respectively the first formalism and the first CASE tool ever developed specifically for spatial databases. Different versions of Modul-R were developed between 1988 and 1995. The versions were supported by a CASE tool called Orion [Pageau and Bédard 1992]. Seven years ago, Orion allowed one to draw a spatially-extended conceptual E/R schema of a desired spatial database, to fill the integrated spatial data dictionary, and to generate automatically the database code for Intergraph’s GIS solution. Modul-R has been used in about ten countries by practitioners and academics. Its spatial component is a de facto standard in Quebec and we argue that it has influenced other spatially-extended formalisms.

In spite of the success of Modul-R, the most recent trends in computer sciences and data modelling motivated us in 1997 to replace this proprietary formalism by the second-generation language presented in this paper. Recognition of the need for a second-generation language emerged from the study of the scientific and commercial literature in conjunction with more than 15 years of involvement with real-world projects. The latter has provided the opportunity to learn important facts about spatial database development, and to integrate pragmatic findings with theoretical principles. The following ten trends summarize our understanding of present and expected situations:

**Trend 1: From relational and E/R modelling towards OO modelling:**

The once dominating E/R modelling paradigm rapidly is being replaced by the OO paradigm. The arrival of UML is one of the main justifications for observing this trend. We note that UML has a tremendous momentum and offers the opportunity to end the multiplicity of formalisms. Note however that this trend applies only for the conceptual level of database modelling. At the physical level of database modelling, the relational formalism continues to be used widely because relational DBMSs continue to dominate the market. We may expect changes in the next 2 or 3 years as users migrate towards the newest RDBMS versions which offer a hybrid object-relational capability.

**Trend 2: From several method-specific OO languages towards a unique standard OO language: UML**

More than twenty popular OO languages were developed at the beginning of the 1990s, each with their specific constructs and graphical notation. This situation has evolved towards a more unified language following the joint effort by Booch et al. [1996; 1999] and other methodologists. This unification through UML rapidly became a worldwide de facto standard. It is now an official standard recognized by the Open Management Group (the main international standardization body for OO).

**Trend 3: From “closed” languages towards extendible languages**

The previous generation of languages had to be used “as is”. Consequently, CASE tools offered no possibility to add new constructs, nor to enrich the formalism for specific purposes (such as spatial
database design). The only exceptions were the metaCASE tools, i.e., CASE tools made to build CASE tools. MetaCASE tools did support several visual languages, and allowed one to modify them, or to create a new one from scratch. However, using metaCASE tools was a very laborious and expensive alternative. UML, on the other hand, has built-in extension capabilities, as does OML (UML's primary competitor). Built-in extension capabilities allow one to include new domain-specific classes of objects, associations, etc. in the formalism while remaining compatible with the core formalism. Such mechanisms can be used, for example, to extend UML for spatial database development.

**Trend 4: From aspatial languages towards spatially-aware languages:**

Spatially-aware languages have become a recent research focus in academia, industry, and standardization bodies such as the OpenGIS Consortium and ISO TC211. A standard visual modelling language would have major impacts on databases designed for spatial querying, spatial analysis, spatial data exchange and system interoperability. A growing interest in spatially-aware languages by the geomatics community over the last three years is an indication of a maturing market.

**Trend 5: From spatially-aware languages towards spatial plug-ins for standard languages:**

This trend follows the general trend in computer sciences towards component-oriented development, software plug-ins for the Web, cartridges and datablades for universal servers. This trend facilitates the addition of specialized modules to core generic software. A plug-in approach is used by UML with regard to extensions, and by the Spatial PVL proposed in this paper. The advantage of spatial plug-ins is that they allow one to keep working with a standard language formalism while adding a simple spatial extension, lowering the learning curve in comparison to learning a new spatial formalism.

**Trend 6: From an expected multiplicity of spatial extensions towards a unique open standard spatial extension:**

Any extension built with UML constructs is not part of the standard UML; only the UML extension mechanism is. Thus spatial extensions would not be part of the UML standard. We consider that it is highly probable that other Spatial PVL will be developed besides the one proposed in the present paper. Hopefully, initial discussions in geomatics standardization bodies (e.g., OpenGIS Consortium, ISO TC211) will result in an official standard extension compliant with UML. This would be a common sense approach.

**Trend 7: From expensive/limited CASE tools towards inexpensive/rich CASE tools and visual modelling capabilities embedded in software development environments:**

In the 1980s and beginning of the 1990s CASE tools were at the same level of complexity and price as market-leading GISs. They were difficult to use and to maintain, sometimes as difficult as the system they described. They were CPU intensive, plagued by restricted import/export limitations, and offered very limited graphics and reporting capabilities. Automatic code generation was not always available, and when available, it was restricted to a small number of DBMSs and programming languages. Nowadays, similar to the GIS trends, excellent packages are available at low prices. They are easy to use, run on small PCs, and are easier to support. The import/export capabilities are more developed and some CASE tools can interoperate from a standard repository. We also find visual modelling tools embedded within software development environments (for example a subset of the Rational Rose CASE tool embedded within Microsoft Visual Studio).

**Trend 8: From “informal and poor dictionary” towards “formal and rich dictionary”:**

This trend results from cognitive research, experimentation, and the arrival of metadata standards for spatial databases. The desire by some researchers to include every geometric detail in the spatial database schema doesn’t correspond to cognitive research results. Diagrams are useful for global analysis and the study of interrelationships (for example for design and validation purposes). On the other hand, textual documents are better suited for specific, detailed information. An analysis of the documents produced by large geomatics organizations confirms this trend. An analysis of the way spatial database developers have used Modul-R is another example. Experienced spatial database developers involved in large, real-life projects don’t put many details into database schemas. They write the details in natural language in the dictionary or in the technical data-acquisition documents. Furthermore, if today’s traditional CASE dictionaries seem poor from a spatial database designer’s point of view, the development of ISO spatial metadata standards and the increased flexibility of UML-compliant CASE tools will facilitate the inclusion of such information into the CASE dictionary.

**Trend 9: From a data-only paradigm towards a multimedia paradigm:**

This trend has two aspects. The first one, which arises mostly from Web applications and universal servers, is to have visual languages supporting the...
modelling of multimedia databases. We need the capability to include in database schema all the pieces of information desired by the client, whatever their media (image, paper document, Web site, video, aerial photography, digital sound or . . . the usual database field). Today’s languages allow only for the modelling of information stored in explicit traditional database fields, not in images, recorded sounds, videos, scanned documents, Web sites, etc. New generations of formalisms, or new extensions to UML, must add this capability as universal servers. Thus, the perfect model leading to perfect code appears to be more a myth than a reality and consequently, today’s database analysts and developers have more realistic expectations about modelling languages and CASE tools; the hype of the early 1990s is over.

With regards to these trends, most spatial-aware visual modelling languages could be considered as first-generation solutions. These first-generation languages are academic method-specific solutions not in sync with the recent trends, nor with the groundbreaking component-based paradigm. Although such a first generation of spatial languages was necessary to build scientific knowledge, its overall limited use by experienced practitioners and academics may indicate usability problems, theoretical problems, a lack of appropriate tools, a lack of integration with mainstream visual languages, or a perceived limited return on investment for their users. It is our opinion that most of these problems apply to most of the existing solutions.

In view of such a fact, and after a decade of academic work, it is time for a new generation of solutions to stem from the developed theory and practical experimentation. The present paper introduces such a second-generation solution. This solution involves two aspects which are presented in the next two sections respectively:

1. applying the generic PVL concept to create a spatial extension for visual modelling languages, that is a Spatial PVL; and
2. integrating this Spatial PVL into a visual modelling language, in our case the UML Class Model, and implementing the result in a CASE tool; called Perceptory.

**An Innovative Solution:**

**The First Spatial PVL**

Spatial PVL stands for Spatial Plug-in for Visual Languages; it is a generic concept created to support the development of second-generation solutions. This section introduces our PVL. It has been in use for 18 months. It has been tested in three large projects. It relies on four strategic orientations:

**Trend 10: From idealistic software engineering practices towards pragmatic modelling:**

This trend has been observed through years of experience. “Every new technology promises to reduce development times and increase success rates of projects, but experienced software engineers tend to be justifiably skeptical of such claims” [Pooley and Stevens 1999]. In fact, modelling is sometimes perceived as an end in itself while it is only a means to an end. This results in over-modelling and overspecifying before programming. We also observe that there is a certain level of detail where experienced systems analysts and designers start having problems in understanding the model. “There is a limit to how much a human can understand at any one time” [Pooley and Stevens 1999]. There is also a limit to what users have the time to formalize within their time and budget constraints. We observed that humans and, surprisingly, the database users, generally have a “soft” understanding of the data they deal with; there are rarely rigorous definitions of objects, attributes, geometries, etc. Humans are used to working with some fuzziness. “It is not always possible or desirable to capture every nuance and restriction in a model: similarly, there is always a danger in producing a convoluted model that captures every small detail at the expense of general understandability” [Booch et al. 1996]. In addition, we noticed the psychological phenomena called satisficing [Davis and Olson 1985] for database analysts and designers who regularly stick with the first database model satisfying them, and who try little to find the very best solution.

Finally, automatic code generation provides usable results only if the database models are very detailed and rigorous; this rarely fits within the above-described reality, and explains why most developers use CASE tools as schema drawing tools only (or to document the system once developed, using backward engineering CASE capacities). Thus, the perfect and true model leading to perfect code appears to be more a myth than a reality and consequently, today’s database analysts and developers have more realistic expectations about modelling languages and CASE tools; the hype of the early 1990s is over.

With regards to these trends, most spatial-aware visual modelling languages could be considered as first-generation solutions. These first-generation languages are academic method-specific solutions not in sync with the recent trends, nor with the groundbreaking component-based paradigm. Although such a first generation of spatial languages was necessary to build scientific knowledge, its overall limited use by experienced practitioners and academics may indicate usability problems, theoretical problems, a lack of appropriate tools, a lack of integration with mainstream visual languages, or a perceived limited return on investment for their users. It is our opinion that most of these problems apply to most of the existing solutions.

In view of such a fact, and after a decade of academic work, it is time for a new generation of solutions to stem from the developed theory and practical experimentation. The present paper introduces such a second-generation solution. This solution involves two aspects which are presented in the next two sections respectively:

1. applying the generic PVL concept to create a spatial extension for visual modelling languages, that is a Spatial PVL; and
2. integrating this Spatial PVL into a visual modelling language, in our case the UML Class Model, and implementing the result in a CASE tool; called Perceptory.
Building on the PVL Concept

The fifth trend presented earlier in this paper indicated how the plug-in approach and the component-based paradigm is changing the panorama of systems developers. This trend also influenced the latest visual modelling languages (e.g. UML and OML with their built-in extension mechanisms). It allows one to add specialized modules to a core generic language in order to fit particular needs. This plug-in approach is the basis of the Spatial PVL concept described here.

From a technological point of view, such a solution is more in sync with today’s open, standard-oriented “not-only-GIS” database development. Consequently, it appears that a plug-in type of solution which is simple and independant from any formalism represents the best scientific and pragmatic solution. Accordingly, the solution proposed in the present paper offers only a spatial extension to visual modelling languages, no underlying new modelling language. This way, users can integrate effectively the proposed solution into the language they already use, or into the one required by the client’s methodology. This lowers the learning curve and facilitates the acceptance of the spatial extension.

To better illustrate this concept, we thought about different terminologies such as ‘cartridge’, ‘module’, ‘extension’, ‘component’, ‘add-on’, ‘add-in’, ‘dialet’ (dialect + applet), etc. We decided to call the proposed solution Visual Languages (Spatial PVL) because the concept and the word are more widely known, i.e. they are known by everyone surfing the World Wide Web, including our target audience. In addition, it conveys very well the basic principle.

PVLs are not pieces of software but rather very small extensions for visual modelling languages. They include basic constructs with a graphical notation and a grammar (usage rules). PVLs can be used manually, in CASE tools, word processors, databases, and so on; some can even be used in natural languages. In fact, similarly to database cartridges, categories of PVL can be developed (e.g. the Temporal PVL and Multimedia PVL included in Perceptory).

A PVL approach provides several advantages. First, the very nature of the plug-in concept ensures that such extensions are intended by design to be small, simple and pluggable to any visual language. Second, it allows one to keep working with the visual modelling language that one knows best. Third, we expect that most PVLs will also be pluggable to non-visual languages and to different categories of tools. Fourth, different categories of PVL can be developed as a family of solutions, and can be 100% compatible among themselves (such as the Spatial, Temporal and Multimedia PVLs in Perceptory), allowing one to simultaneously use complementary PVLs with a minimum of training and problems. Finally, PVLs are used only when needed.

Nevertheless, like any language, the PVL constructs, graphical notations and grammar must be learned. In spite of their simplicity, they must be mastered to be used properly in more complex situations. Furthermore, some software tools may have no possibility to use certain PVLs, either as a result of a limited tool or of a poor PVL design. Finally, PVLs are additions to more complete visual languages, thus the quality of the design of a spatial database depends first of all on the mastering of the full language; the PVLs do not ensure good models, they simply make them easier to build, edit and read (whether the models are good or bad), although they may include some constructs and rules facilitating the validation of a model.

Symbiotic Approach

From a scientific and practical point of view, when one thinks of an efficient visual modelling language, one must take into consideration the complete HLS modelling environment (Human + Language + Software). Developing a language in isolation of the other components of the modelling environment may result in an interesting technical solution; however, it may fail to be used properly by humans or it may be supported inadequately by software tools. In other words, new solutions must be built around the symbiosis between the visual modelling language...
and its users into their context and with their tools. Such considerations refer to a more complete scientific approach when one remembers that the limitations related to modelling always relate to both the modelling process (including tools) and the modellers (including context) [Bédard 1987]. Interestingly, this approach has led our PVL towards simplification and a certain elegance.

Several research methods can be used to develop a solution supporting this symbiotic HLS approach. One way is to analyse what people understand and use from existing solutions within a typical development context. Another way is to try different variations of the proposed language, and to select the most effective ones. A third way is to test with large and complex database schemas, and see how people do these or react to them. A fourth way is to present models and dictionaries to a variety of people, from programmers, to application experts, to their bosses, and to study their reactions and level of understanding. A fifth way is to model spatial databases in very diverse fields of application, and to see how the language can express different categories of situations. A sixth way is to try modifying a year-old database model, which we have done, and to identify the difficulties encountered. A seventh way is to try modifying a database model made by someone else, identifying the difficulties in doing so.

We have used all seven research methods presented above on an empirical basis, relying on informal observation during and after experimentation. These experimental observations and modelling tests included large and complex spatial databases such as the Canada National Topographic Database (its first and second generations), Quebec Provincial 1:20 000 Topographic Database, Quebec Provincial Road Network Database, British Columbia Road Network Database, Quebec Provincial Official Administrative Boundaries Database, Quebec Provincial Topographic and Administrative 1:250 000 Database, Montmorency Forest database, and Eco-Recherche Environmental Database as well as smaller projects in climatology, urban facilities management, transportation security, cadastral, property assessment, and spatial data cataloguing with metadata. Such projects involved multiscale considerations, multirepresentation and generalization, Web distribution, metadata, alternate geometries, facultative geometries, complex objects, automatic updating, temporal issues, topological relationships, and spatial integrity constraints. Over the years, the technologies involved included Oracle, Informix, Access, DBase, FoxPro, Intergraph TIGRIS, Intergraph MGE, Intergraph Geomedia, ESRI ArcView, MapInfo, PAMAP, Intergraph Geomedia WebMap, and Autodesk MapGuide as well as visual modelling tools like Oracle Designer/2000 and ObjectMaker, plus diagramming tools such as Visio and FlowCharter.

This variety of experiences occurred within governmental, private and academic projects. They have strongly influenced our choices in the selection of the basic constructs of the proposed Spatial PVL, in its level of detail, in its graphical notation, and in its balance between graphical and textual information.

Regarding this latest issue, we relied on research in cognitive sciences which has shown that graphics help to develop global views and to show interdependencies between the elements of a system, while texts are better for specific details. Cognitive sciences also have shown that combining both graphical and textual languages is necessary to achieve a clear understanding of a topic. Nevertheless, the difficulty when designing a PVL is to decide which parts of the model should be presented graphically, which parts should be described textually, and where there should be overlap. Several possibilities exist. We must look for the highest richness of expression necessary to simplify the modelling of a spatial database while keeping it easily readable.

In the Spatial PVL proposed in the present paper, the balance between graphical and textual is set to facilitate the drawing of the database schema and the integration of the PVL into commercial CASE tools. This strategic choice led us to keep the graphical information to a minimum, and to insert specific details, spatial integrity constraints and exceptional situations in a textual form into the dictionary. This should also encourage the acceptance and use of the Spatial PVL. One must realize that the graphical notation is often and mistakenly perceived as the entire language, and if it is not accepted, the rest of the language won’t be. As stated by Pooley and Stevens [1999], “the chosen language should be . . . easy enough to use, so that the modelling language aids clear thought rather than getting in the way”. Such simplicity is a key issue for the acceptance of the proposed solution.

Adhering to Major Trends

In addition to the previously noted trends 3, 5, 8 and 10 which are at the centre of the Spatial PVL developed, other trends contributed to deliver a better solution. Some trends influenced, to a lesser degree, our Spatial PVL (trend 9) while others contributed to our CASE Perceptory (trends 1, 2, 7, 9). For us, it was important to identify and work towards these trends, which we expect to shape the future of spatial database modelling. This allowed us to figure out the characteristics of the second-generation solution we
were looking for, and to stop investing in trend 4 (spatially-aware languages) which seems to have a rather difficult future.

**Higher Level of Abstraction**

The Spatial PVL proposed in this paper has been developed for ‘data models’ and ‘object class models’ as these are the primary models used to design databases in the E/R and OO paradigms respectively. We also decided to focus on the conceptual level since this is where we found the greatest challenges and the highest needs to facilitate the spatial database modelling process. In other words, we wanted to facilitate the analysis phase of spatial database development by supporting a slightly higher level of abstraction than usual in order to better reflect the users’ perceptions of their reality. From our point of view, this is an area where non-spatial languages fall short on efficiency for spatial databases.

We have observed that people often have difficulties in understanding database models, for example the clients’ application specialists who participate in the so-called Database Technical Committee. We also have observed that database analysts and designers rarely master the full power of a visual language. They frequently limit themselves to the basic constructs because they find the overall language too complex or they don’t see the utility of using all its constructs.

Facing such a reality and in accordance with our symbiotic approach, we targeted the famous 7±2 psychological threshold, i.e. humans can only master 7±2 elements at once [Miller 1956]. In accordance with this principle, the developed Spatial PVL includes only the key constructs in the graphical notation and hides several complexities in the dictionary (for example the intricacies of lines vs. oriented lines vs. polylines, and metric vs. topology, spatial integrity constraints, complicated shapes). The proposed Spatial PVL also eliminates the explicit graphical representation of rarely met aspects (e.g. 3-D objects).

In spite of such simplifications to encourage higher levels of abstraction, the Spatial PVL remains expressive enough to present clearly the key geometric information relevant to the application specialist:

- does the user want to have this class of object represented on the map?
- if yes, using what geometry?
- is this geometry derived from other geometries?

**The Developed Spatial PVL**

Several possibilities exist for developing a Spatial PVL. The Spatial PVL presented in this paper has a small set of basic constructs (3) with a minimum number of variations (7). Nevertheless, it has a very high richness of expression. These constructs and variations are presented in Figures 2 and 3 respectively.

The pictograms of our Spatial PVL are designed to be inserted into the database schema. They may be inserted at different places for different visual languages, for example within the box of an entity type or object class on either side of its name, or at either the top or the bottom of the box. The Spatial PVL constructs could also be represented by geometric fields instead of pictograms (dimension, combination, derivation, cardinality), or in a geometry section under the method section of object classes. In fact, this Spatial PVL allows some degree of liberty with the graphical notation

---

**Figure 2:** The three basic constructs of the developed Spatial PVL with their graphical notation (called ‘pictograms’).

<table>
<thead>
<tr>
<th>pictograms</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>0-dimensional shape (example for hydrants when they are all represented by a point)</td>
</tr>
<tr>
<td>L</td>
<td>1-dimensional shape (example for road segments when they are all represented by a line)</td>
</tr>
<tr>
<td>P</td>
<td>2-dimensional shape (example for lakes when they are all represented by a polygon)</td>
</tr>
</tbody>
</table>

**Figure 3:** The seven variations of the Spatial PVL with examples of graphical notation.
as long as it remains consistent with the constructs. This choice is strongly influenced by the limitations of the CASE tool being used. In our own implementation of the Spatial PVL, we place the pictograms on the left side of the name of object classes having a geometry. We also use the pictogram for the attributes which vary spatially within classes having a geometry. We also use the pictogram on the left side of the name of an object implementation of the Spatial PVL, we place the appropriate symbols within the pictogram box.

The following paragraphs explain the details of the Spatial PVL.

**Simple geometry:** As indicated in Figure 2, we insert a spatial pictogram when the client wants to have the position and shape of an object class on the map. Most of the time, an object class has a simple shape which is 0-D, 1-D or 2-D. For example, the shape of every tire hydrant may be a point, the shape of every road segment a line and the shape of every lake a polygon. In this case, the tire hydrant object class will have the 0-D pictogram (Figure 2, line 1), the road segment the 1-D pictogram (Figure 2, line 2) and the lake the 2-D pictogram (Figure 2, line 3). This means that every instance of those classes will have exactly one instance of such a geometry.

With regards to three-dimensional objects, they are represented by their orthogonal projection on the surface of the earth and the 3-D details are described textually in the dictionary. The assumption is that three-dimensional objects practically never occur in spatial databases.

**No geometry:** One must remember that not all classes of objects have spatial pictograms, only those which have an explicit geometry have pictograms (see the entity type PERSON in Figure 1).

**Facultative geometry:** If the shape of an object instance is facultative based on certain criteria, then we add the 0,1 cardinality next to the pictogram (Figure 3, line 3). For example, a building may be given a geometry only if its area is larger than 1 hectare, otherwise it is a non-spatial object. One must know here that the default cardinality for simple shapes is 1,1 - and that we do not write it.

**Simple aggregate geometry:** The geometry of a class of objects may also be an aggregation of shapes. If these shapes are of the same dimension, we simply add the 0,N or 1,N cardinality next to the pictogram. For example, road networks are composed of several lines, consequently the Road Network class has the 1-D pictogram followed by a 1,N cardinality (Figure 3, line 3).

**Complex aggregate geometry:** If the aggregation involves shapes of different dimensions (i.e. a complex geometry) this can be depicted by inserting the appropriate symbols within the pictogram box. For example, a hydrographic network is an aggregation of several lines (rivers) and polygons (lakes) and consequently its pictogram box includes a 1-D and a 2-D symbol (Figure 3, line 1). This pictogram box is not followed by the cardinality 1,N because we decided to make the 1,N cardinality the default for complex shapes. (In fact, we have never encountered a complex shape with a 1,1 cardinality).

Nevertheless, there could be other cardinalities.

**Alternate geometry:** When each occurrence of a class of objects has a shape of either one dimension or the other but never both, then we use the alternate pictogram. The alternate geometry is represented simply by the concatenation of the possible pictograms (Figure 3, line 2). There is no space between the pictograms and their ordering has no meaning. For example, when a building is represented by either a point if its area is smaller than 1 hectare or by a polygon if its area is larger than 1 hectare, then it has the 0-D and 2-D pictograms adjacent to each other (see Figure 1 for this example). This is the exclusive ‘OR’ as opposed to the aggregated shapes which represent the ‘AND’.

**Any possible geometry:** If a class of objects may have any combination of shapes, then we use the wildcard pictogram (the asterisk, Figure 3, line 5). For example, an object class such as ‘Historical Feature’ may be a point (e.g. a statue), a line (e.g. an historic street), a polygon (e.g. a park), a polygon with a line (e.g. an historic place including adjacent streets), or a set of disjoint polygons (e.g. a group of buildings). We use this wildcard when we assume that every geometry is possible, without restrictions.

**Complicated geometry:** If an object has a geometry which is too complicated to express with the pictogram, or a limited set of possible geometries, then we use the complicated shape pictogram (the exclamation point, Figure 3, line 6). A first example is the object class ‘Bus network’ which is represented by a complex aggregation of lines (road segments), points (bus stops) and polygons (parking). It can be considered a complicated geometry, and can be represented by the exclamation pictogram. A second example is the object class ‘Property’ which could be a polygon (a lot), a point or a polygon (a small or large building), a line (a single private street), or an aggregation of lines (a small private street network). We use the exclamation pictogram when it becomes a tricky job to indicate graphically the geometry of an object class. We have a rule of thumb suggesting the use of this pictogram when more than 2 pictograms are required to describe the proper geometry of an object class. We then use the database dictionary to explain all the possibilities and limitations.

**Derived geometry:** Sometimes, the shape of a class of objects may be derived from the shapes of other classes of objects (county polygons are
GEOMATICA

derived from municipal polygons). The shape may also be derived from another shape of the same class (a building centroid used for small-scale maps can be derived from the large-scale polygon for the building). Such derived shape is represented by italicized pictograms (Figure 3, line 7). The derivation rule must be described in the dictionary. All types of geometry may be derived.

If we can derive the shape of only a part of the occurrences and need to digitize/measure the others, then we still use the italicized pictograms but we indicate the proper information in the dictionary. This happens frequently for large territories. For example, we could have large-scale digital maps with building polygons only in the South of the province, then digitize building centroids from small-scale paper maps in the North.

Multiple geometries: The client may also require that a class of object has more than one shape for each occurrence. Such multiple shapes may happen when a second shape of an occurrence of object is derived from its first shape (the buildings represented by a centroid and a polygon). It may also happen when a second shape is collected in addition to the first shape of the same occurrence of object (municipalities represented by a polygon and by a point located downtown, such point being non-derivable and different from a centroid). There are no limits to the number of different shapes an object class or an attribute may have. To indicate such multiple geometries, we simply insert all the needed pictograms (Figure 3, line 4) with their cardinality if necessary. However, we insert a space between these pictograms to differentiate the multiple shapes from the alternate shapes (which use adjacent pictograms). Multiple geometries usually are required for multiple representations at multiple scales.

Geometry of attributes: So far, we have used the spatial pictograms for the geometry of object classes or for entities. However, it may happen that we want to describe the spatial distribution of an attribute within an object without splitting the object into sub-objects (at least, from the point of view of the client). For example, in transportation, it is a common rule to use a linear referencing system for certain road attributes such as “number of lanes” (starting from intersection A, the road has 4 lanes up to a distance of 200 m; then 2 lanes from the distance 200 m to the distance 1800 m). When the client requires such spatial distribution of an attribute within an object class, we simply use the previously presented pictograms next to the name of the

Figure 4: Inserting the geometry ‘Area’ into the ‘Lot’ object class with Perceptory.

171
attribute. This indicates that we will insert in the databases all the data necessary to know where the attribute changes its values within an object. This also is useful for attributes with a continuous spatial distribution within an object (for example elevation data changing at every point within a polygon).

Since we use the Spatial PVL at the conceptual level, the client doesn’t have to see how we will implement the spatiality of an attribute (for example the creation of sub-objects vs. dynamic segmentation). After all, this implementation depends on the technology used, and on programming strategies, but it should remain as transparent to the client as possible.

Finally, the Spatial PVL suggests including the details of a database schema into textual information within the dictionary. In other words, one can include in the dictionary all the information needed to complete the key constructs provided by the graphical notation. Consequently, the content of this dictionary can be very rich. It provides the opportunity to include many ISO TC211 spatial metadata, and to be more precise about spatial definitions, semantics, and constraints (see extracts in Figures 6, 7 and 8). Although the level of detail is left to the user, minimum detail must be specified to validate a database model. This minimum includes the semantic definition of object classes, attributes, functions and relationships. It also includes the spatial definitions of the object classes and of the attributes, the attribute domains, the function semantics and all derivation rules.

A formal dictionary provides a structured way of entering detailed information and producing rich reports. Without being exhaustive, let’s say that the dictionary of our Spatial PVL includes definitions of feature types, feature attributes, feature functions and feature relationships (using ISO TC211 vocabulary). The proposed graphical and textual components of such a Spatial PVL have already been used in real projects. Based on a comparison with non-spatial formalisms published by Bédard et al. [1996]
GEOMATICA

concerning the ancestor of our Spatial PVL (Modul-R spatial language), the use of pictograms simplifies the visual modelling of spatial databases in the order of 40% for OO and 65% for E/R models. In addition, this Spatial PVL facilitates the thinking process and the reading of the database models, leading to easier discussion with clients and programmers. Such results are among the most important benefits of this research since they lead to a more efficient spatial database development process.

Finally, the rules to implement a conceptual model built with the proposed Spatial PVL depend on the specific software used. In general, it is rather straightforward (it has been done for recent projects using ArchInfo with Oracle, Access with Java on the Web, Intergraph MGE with Oracle, Intergraph Geomedia and MapInfo). Such formal rules also are being implemented in an automatic code generator for Oracle 8 Spatial Cartridge. This will be included in the in-house developed freeware CASE tool presented in the next section.

Perceptory: An Innovative UML-Based Visual Modeling Tool with PVLs

Using the proposed Spatial PVL, one can improve to a certain point today’s commercial CASE tools for the development of spatial databases (as we do for teaching purposes; see Figure 1). However, there is also a need for researchers to develop their own solutions for R&D purposes and further innovations. In this section, we introduce such an in-house visual modelling tool called Perceptory. It is an innovative solution plugging our Spatial PVL into the Unified Modeling Language (UML) Class Model.

This CASE tool was developed for high-level conceptual class models of spatio-temporal databases with multimedia capabilities (this paper presents the spatial aspect). From a technical point of view, Perceptory extends the UML language using UML’s

Figure 6: Perceptory user interface for the database dictionary (Semantics tab).
Ilie

Ensemble : Réseau hydrique

Définition : "Terre entièrement entourée d'eau" (DEGQ, 98).

Référence spatiale

Pictogramme : [Image]

Règle géométrique : L'île se compose de primitives d'île

Échelle 1/250 000 et 1/20 000:

Une île est toujours située dans un lac ou un cours d'eau superficiel.

Surface : > 00 m²

Règlement : La géométrie est dérivée de la primitive d'île &.

Détails :

Figure 7: Perceptory default output for the database dictionary.

'stereotype' construct (Booch et al. 1999). However, Perceptory has a special way of presenting these stereotypes to the user, a way which is easier to implement and to use when a large number of spatial stereotypes is needed (in our case, one for each pictogram, each combination of pictograms, each PVL). Figure 4 shows how to add the geometry 'Area' to an object class (Lot) by using the 'Area (simple)' command; this command inserts the 2-D pictogram into the object class box, on the left side of the name 'Lot'. Similarly, one could select from the mouse right-button pop-up menu the command to insert the 0-D or 1-D simple pictograms, or to insert 'Other' less frequent pictograms (Alternate, Complex, etc.) or even the 'Complicated' and 'Wildcard' pictograms. This is a much simpler user interface than what can be implemented with extended commercial CASE tools.

This same pop-up menu also is used to insert attributes and operations into the object class box or to enter and read the content of the dictionary. Figure 5 shows an example of a spatial database schema designed with Perceptory while Figure 6 gives an idea of the details of the dictionary for an object class. There is one form to fill for each tab: Semantics, Spatial definition, Temporal definition, Spatial evolution, Media, Attribute, Functions. Similar forms are found for each attribute, each
domain, each function and each relationship. Figure 7 presents a normal output for an object class while Figure 8 shows an ISO output. This content is a superset of ISO TC211 (which are written in a different colour in Figure 5).

Perceptory complies with the major trends and strategic choices identified earlier. It adds a few non-spatial graphical notations as well as a Temporal PVL and a Multimedia PVL. Perceptory is a free template developed for VISIO Standard (www.visio.com), it is downloadable from http://sirs.scg.ulaval.ca/yvanbedard. It was developed as a VISIO template for several reasons:

1. VISIO Standard’s low cost makes it affordable for students;
2. it is programmable in Visual Basic for Application instead of a proprietary language;
3. its database engine is the same as Access (thus directly accessible via Access for extra querying and reporting flexibility);
4. it is a full and simple MS-Office compatible graphics package already used for general purposes; and
5. it is highly popular and a very flexible solution allowing us to highlight, colour, modify or add to the database schema whatever graphical element or note we want.

The integrity constraints of the database are written in the dictionary. These constraints can be written in the Object Constraint Language (OCL), [Warmer and Kleppe 1998], in natural language (such as English or French) or with any other language since it is stored in a text field. The spatial integrity constraints are managed by a specific module. Following testing in real projects with an

---

Figure 8: Perceptory IS0 TC211 output for the database dictionary.
GEOMATICA

Figure 9: Prototype module to enter, in the dictionary, the spatial integrity constraints between two object classes.

Discussion and Conclusion

We find that visual modelling languages and their CASE tools offer an elegant solution to support a spatial database development process. While mastering a modelling language is necessary to develop complex databases and to design efficient systems, mastering a CASE tool is not as critical to the success of a project. Pouliot et al. [1997] have described different visual modelling tools for the development of spatial databases, including non-CASE tools. For example, there are the simple and inexpensive drawing packages (such as CorelDraw) and traditional diagrammers (such as VISIO Standard and FlowCharter). The latter help to draw the database schema, but the dictionary typically has to be written into a word processor. However, there are no links between the schema and the database, leading to possible inconsistencies. On the other hand, these packages are widely available and easy to use. They present a minimal solution for the casual users. Perceptory has been developed to run in such packages while overcoming their limitations.

For the more advanced users, there are the traditional commercial CASE tools. Some CASE tools support only one modelling language while others support several. Most provide automatic
code generation for SQL and popular programming languages (but not for spatial databases engines nor GIS). These are the traditional CASE tools used by experienced database developers. However, one needs to extend such tools with a Spatial PVL to increase the efficiency of analysts and programmers. For example, we extended Oracle Designer/2000 with our Spatial PVL by adding a special font with our pictograms (available at http://sirs.scg.ulaval.ca/yvanbedard) and by extending the dictionary (Oracle Repository). The schema shown in Figure 1 was done with such an extension, and this solution has the advantage of being available from a commercial product with minimum addition. However, such a solution is not as smooth and elegant as Perceptory since extension capabilities of traditional CASE tools are often quite limited and the learning curve rather steep.

Another and similar solution would be to expand on CASE tools by replacing the pictograms by an object class naming convention and by adding attributes in the dictionary to specify the dimension, combination, cardinality and derivability of the geometry. This is perhaps the most standard and universal way of proceeding. However, it would be complex and confusing to use this solution for the attributes as well as for the object classes and their associations. Also, such a solution is not as intuitive nor as complete as what is offered by Perceptory. Only highly extendible CASE tools could provide a serious solution of this type. Nevertheless, this is a solution we will investigate more seriously with the new breed of CASE tools based on UML.

The last possibility is to build your own CASE tool. Here one has the choice between different solutions: first to develop everything from scratch with a programming language like C++, second to build on top of a software component, third to program a diagramming tool such as VISIO, and fourth to code a metaCASE tool (i.e. a CASE tool built especially to build CASE tools). Up to two years ago, the latter solution was the best and only practical one. It was the most widely used solution by large organizations having their own in-house development method. This is also what we did seven years ago with Modul-R see Pageau and Bédard [1992] and what we suggested in publications written prior to the Fall of 1997 (see for example Bédard [1999] but remember that the manuscript was written in the Spring of 1997).

Since the integration of VBA within VISIO, we have had access to VISIO objects and we could develop a rich solution with a minimum investment in programming. We found several other advantages to switching from an expensive/proprietary/rigid solution to an affordable/open/flexible solution. They are readily available to students and the larger geomatics community, and have a simple learning curve, reusability of tools for all kinds of drawing besides database models, reusability of tools for several visual modelling languages in the
Professional version of VISIO, and even complete CASE tools in the Enterprise version. This explains why we changed our strategy and opted for a somewhat surprising solution: programming VISIO with VBA. The results of the last 18 months have reinforced our position.

Nevertheless, CASE tools are not the most important part of spatial database design. Successfully developing spatial databases requires the mastering of a visual language to understand the users’ needs and to master the design of robust solutions. To this end, we have enriched traditional languages by developing a Spatial PVL. Created from theoretical and empirical findings, this visual language extension has been tested in real projects. Integrating the proposed Spatial PVL into one’s modelling language provides additional guidance and rigour to the analysis, design and programming of a spatial database. It better supports the thinking and technical processes as well as encourages consistent communication and documentation.

Finally, an implementation of this solution in a PVL-oriented UML-based free VISIO template called Perceptory represents an innovative solution. Our first studies and experiences suggest that the proposed Spatial PVL as well as Perceptory improve the development process when compared to first-generation solutions. In addition, their effectiveness and simplicity help to overcome the limited popularity of first-generation solutions. The introduction of a second-generation solution appears very timely when one observes today’s convergence of spatial database management technologies with mainstream technologies and system development methods.

References


Coad, P. and E. Yourdon. 1991a. Object-Oriented Analysis, 2nd Ed. Prentice Hall.


Miller, G.A. 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for...


**Acknowledgments**

The author wishes to thank the Natural Sciences and Engineering Research Council of Canada for financing this research. The author also wishes to thank the Ministry of Natural Resources of Québec, and Natural Resources Canada (CITS-Geomatics Canada) for their contribution to our most recent projects, as well as the following graduate students and research assistants for their participation in the design and development of Perceptory: Suzie Larrivée, Marie-Josée Proulx, Stéphane Daudelin, Christian Mattel, Clément Nolette, Pierre Normand. Finally, the author also thanks the anonymous reviewers for their valuable comments and suggestions.

**Author**

**Dr. Yvan Bédard** is a professor at Laval University Geomatics Sciences Department, since 1986. He teaches GIS, Spatial Database and System Development courses. He was the Founding-Director of the Centre for Research in Geomatics (CRG) and the Scientific Director of the Centre for the Development of Geomatics. He is an active member of the CRG and of the newly created Canadian Network of Centres of Excellence in Geomatics (GEOID) headed at Laval. Dr. Bédard has been involved in several major R&D projects, often with industry and governments, in Canada and abroad. He has presented papers in almost 300 journals and conferences.