To appear in: Proceedings, 1998 IEEE Symposium on Visual Languages, Halifax, Canada, Sep. 1-4, IEEE Computer Society Press, 1998.

VISCO: Bringing Visual Spatial Querying to Reality

Michael Wessel and Volker Haarslev
University of Hamburg, Computer Science Department,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
http://kogs-www.informatik.uni-hamburg.de/~haarslev/
http://kogs-www.informatik.uni-hamburg.de/~mwessel/
Please visit the VISCO homepage at
http://kogs-www.informatik.uni-hamburg.de/~mwessel/visco-engl.html

Abstract

This paper reports on the evolution of the spatial (sketch-based) query language VISCO and its implementation. The first design of VISCO's query language was presented at VL '97. The language is based on a strong naive physics metaphor for query objects (e.g. marbles, nails, rubber-bands). We shortly review the prominent aspects of the revised version of VISCO's query language. The main focus of this paper is on VISCO's implementation using city maps of Hamburg as example domain. Its innovative user interface consists of three interconnected components: a graphical (syntax-directed) query editor and visual language compiler, a browser for inspecting the query results, and a map viewer for browsing the spatial database. We also briefly report on the process of compiling, optimizing, and executing VISCO's queries.

Keywords— visual query languages, environments and systems, graphical representation of constraints, human computer interaction (HCI), spatial information systems (SIS), graph matching, optimizing visual language compilers.

1 Introduction

At VL '97, we presented the first design of the visual (sketch-based) spatial query language VISCO [1]. Continuing our work, this paper reports on a first prototypical implementation of VISCO that integrates several components to form a visual query system. We will mainly discuss the user interface which is composed of three main components: the *graphical query editor* offering "sketch"-based querying of spatial databases (e.g. containing digital vector maps of the city of Hamburg), a browser-like *query-result inspector*, and a powerful *map-inspection tool* for the spatial database. Another component of the system is the optimizing visual language query compiler. Our prototype successfully demonstrates the usefulness and feasibility of VISCO and its underlying language concepts. The current prototype is fully implemented as described in this paper.

According to the point of view of many other authors in the field, we assume that the term *visual language* denotes a means of communicating with a *visual system* in a coherent and consistent way through *visual expressions* (e.g. see [2]). In our opinion, a visual query language should not be considered in isolation, but in an integrated environment providing an easy-to-use visual query system, offering active support and feedback, strong metaphors etc. In fact, it often becomes obvious that the usefulness of a visual language heavily relies on an appropriate interface which therefore – at least for the user– *is* the language (see [3]).

The remainder of this paper is structured as follows. Section 2 briefly reviews the prominent aspects of VISCO's revised visual query language [1]. Please note that the visual appearance of the language has been slightly changed but not the underlying key concepts. Section 3 reveals the logical architecture of the system and describes the graphical user interface (GUI). Section 4 focuses on important aspects of the implementation with respect to the design of visual language compilers and reports on the abstract syntactic representation, the optimizing compiler, and the process of query execution.

2 The Visual Query Language VISCO

VISCO is a visual spatial query system designed for extracting information from spatial information systems (SIS, especially GIS) in a visual way. The term spatial information system refers to a broad class of systems which collect, manage and offer the analysis and presentation of spatial data (e.g. lakes, roads, and buildings in a GIS). These spatial objects usually have *spatial* (e.g. a lake is a polygon) and *non-spatial* or *thematic* aspects (e.g. an object is of type "lake"). In its current stage of development, VISCO is primarily targeted for querying spatial aspects of GIS data in a visual way. Currently, no complex thematic statements (e.g., attribute "joins") can be expressed; we only support simple thematic "typing" of query objects (like "city"). Integrated spatial query systems or environments can be considered as a way to facilitate a number of severe problems and limitations found in conventional non-spatial query languages (e.g. extended SQL) with respect to spatial aspects of the data (see [5]). Therefore, we focus on particular spatial aspects: *VISCO supports the retrieval of interesting constellations of spatial objects based on their structural, topological, metric and geometric attributes and relationships between them.*

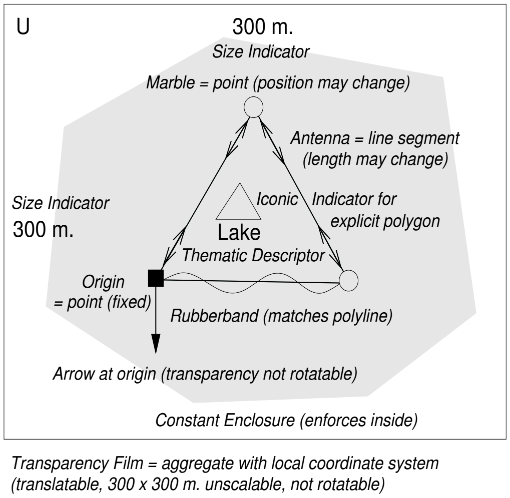

Figure 1: A simple VISCO query (annotations in italic font)

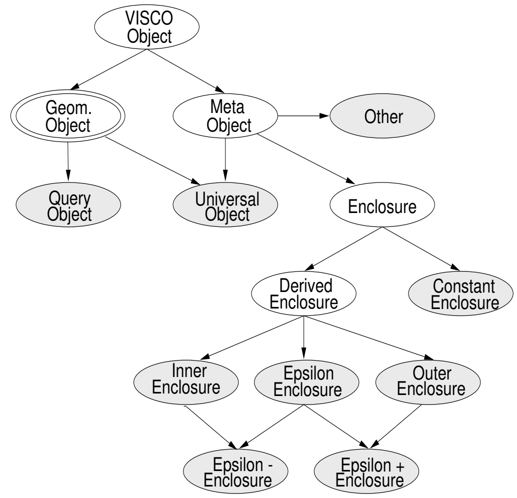

Figure 2: Query language elements supported by VISCO (non-shaded nodes represent auxiliary concepts; a refinement for the node "Geom. Object" is given in Figure 3)

As a first example, the simple query shown in Figure 1 could by described as "Search for a lake of (nearly) arbitrary form that is not bigger than $300 \times 300\,m$.". The semantics of the elements used in the query are explained in the figure by annotations.

In the following, we assume a topologically structured vector representation of the data of interest. Data models like the one assumed here can be found in advanced vector-based GIS (in contrast to raster-based GIS). The data in vector-based systems usually consists of nodes or vertices (points), edges (lines) and faces (simple polygons). Additionally, polylines and arbitrary aggregates of these objects can be found. The "direct component of" relationship between these objects forms a DAG (Directed Acyclic Graph). In our case, the DAG has a maximum depth of 4 because aggregates may contain polygons (but never other aggregates), which are built from lines, and lines contain their end points. Together, object classes and the operations on them form the logical data model of the spatial information system. Like SQL, which is only suitable for the relational data model, VISCO can only be linked with (topologically structured) vector-based SIS.

The following query language elements are handled by VISCO (see Figure 2):

VISCO Objects: A *VISCO object* is any element of the visual language VISCO.

Geometric Object: *Geometric objects* are points, lines, simple (non self-intersecting) polylines, simple polygons and aggregates (see Figure 3). We distinguish two types of geometric objects: *query* objects and *universal* objects (see below).

Query Objects: A *query object* is a geometric VISCO object that matches geometric objects in the spatial database. We can also say, that a query object *represents* a database object. Geometric database and geometric VISCO objects must be identically structured or very similar (in a way discussed above) because VISCO's query execution is based on *graph matching*.

Universal Objects: Unlike geometric *query objects*, which must directly match objects in the spatial database, a *universal* or *auxiliary object* represents an object in the universe of all well-formed geometric objects. Universal objects are primarily used for expressing additional constraints on query objects, and therefore are considered as *meta objects*. The prototype requires, that universal objects can be instantiated by other objects (e.g., as an operator result or through its component objects – a universal segment could be instantiated by its end points provided they are query objects).

Meta Objects: A *meta object* is a VISCO object that visualizes some additional conditions (constraints) on other VISCO objects and therefore makes statements *about* these other objects and their interpretation. Special meta objects are *enclosures*, other meta objects visualize other possible constraints (in form of arrows, text objects etc).

Enclosures: An *enclosure* is a meta object representing a (connected) subset of R^2. We distinguish *constant* (or *sketched*), *interior* and *exterior* (for polygons), and $\varepsilon(r)$ enclosures. Enclosures may be *translucent* or *opaque*. Opaque enclosures are used to (partially) occlude other objects and to (partially) disregard their existence. With the help of opaque enclosures one can express spatial "don't cares" by ignoring spatial relationships with occluded objects. In order to compensate for this visual incompleteness, VISCO's language has to be integrated into a supporting environment because it might be impossible to reconstruct the

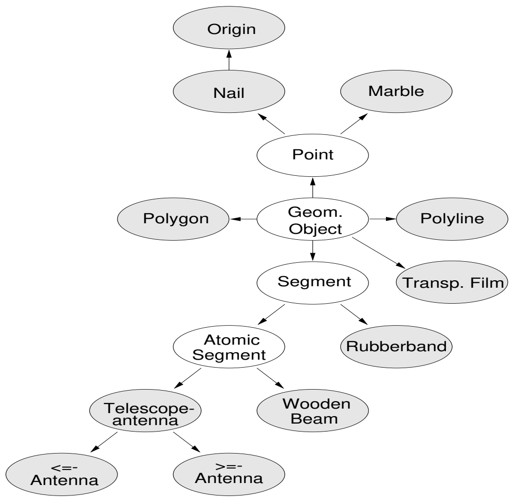

Figure 3: Geometric objects of VISCO

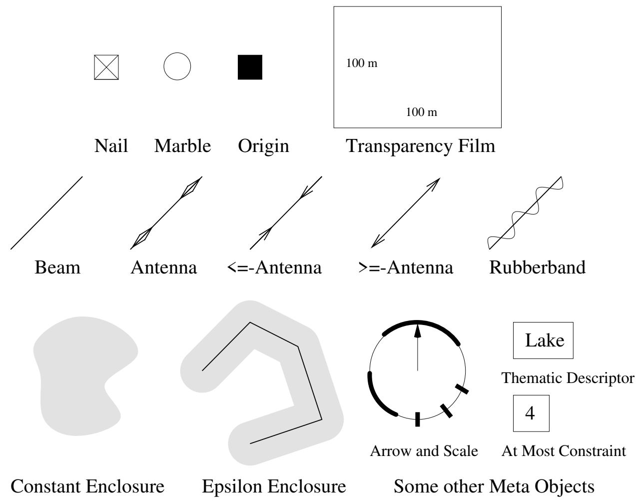

Figure 4: Visual appearance of various VISCO objects.

semantics of a query only from the final drawing.

Derived Objects: *Derived objects* comprise interior, exterior, and $\varepsilon(r)$ enclosures, but also derived center points, calculated intersection points, etc.

We associate each geometric VISCO object with a metaphorical naive physics semantics that is intended to guide the user's interpretation of spatial aspects in a visual representation, e.g., a marble can roll around and change its position in contrast to a nail (see Figure 3). Therefore, the meaning of the aspect "position" of a visualized point object can be immediately discovered by a user because of the attached "naive physics" semantics. We use the following metaphors (please note that geometric VISCO objects are divided into query objects representing database objects and universal objects representing objects in the universe of geometric objects, see Figure 2):

Transparency films represent transformable aggregates with a local coordinate system. Transparency films can be translated, rotated, scaled, and stacked upon like layers. Each geometric object (except transparency films) must be defined on exactly one transparency film (the so-called *carrier*). Special constraints regarding scalability and rotatability of films can be established (e.g., a film is defined as unscalable and has an extension of $100 \times 100\,m$ or may be scaled only proportionally, etc).

Nails and marbles are special points: a nail on a film represents a point with a quantitative (exact) position (relative to the local coordinate system of the carrying film), but a marble can roll around in an enclosure (see below) and therefore it has a qualitative (or vague) position. A marble has to be inside of at least one enclosure. The origin of a transparency film plays the role of a special nail.

Rubberbands, telescope antennas and wooden beams are special lines. A *telescope antenna* (also referred to as *atomic rubberband*) represents a straight line with arbitrary length. A '$\leq$-*telescope antenna*' represents a straight line with a given maximal length (relative to the metric of the carrying film's local coordinate system). A '$\geq$-*telescope antenna*' represents a straight line with a given minimal length. A *wooden beam* represents a straight line with an exactly known (quantitative) length. A general *rubberband* represents a polyline of straight lines (slightly simplified).

The visual appearance of various VISCO objects is shown in Figure 4. The element "Arrow and Scale" is used for establishing orientation constraints for origins (regarding the rotatability of the whole transparency) and for beams or antennas (regarding the orientation of the element w.r.t. to the carrying transparency). Due to lack of space we can not discuss some other elements (e.g. the angles allowed between beams and/or antennas can be restricted with an element similar to "Arrow and Scale").

Constructing a VISCO query is a progressive process: at the time when a new object is created, various high level topological (spatial) constraints between the new object and already existing objects are established. Here, the notion of (partial) visibility becomes crucial (see below). Also, each component object (e.g. a segment of a polygon) is considered as an individual object with its own identity – however, topological constraints involving components can be discarded if necessary, yielding more relaxed queries. The following topological relationships between a newly introduced object and any existing object that is not totally occluded by an enclosure are recognized by VISCO:

- **Disjoint** is established, if the other object is completely visible.
- **Intersects** is established, if the new object has at least one visible intersection point with the other object.
- **Inside** is established, if the new object is completely

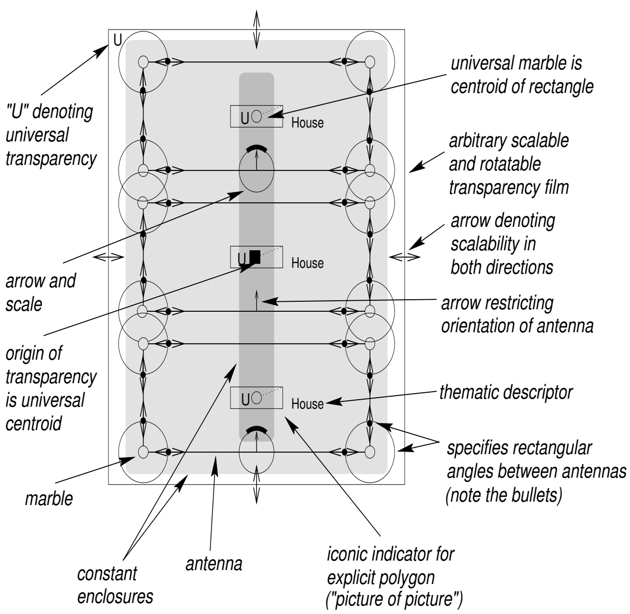

Figure 5: A query for searching 3 buildings.

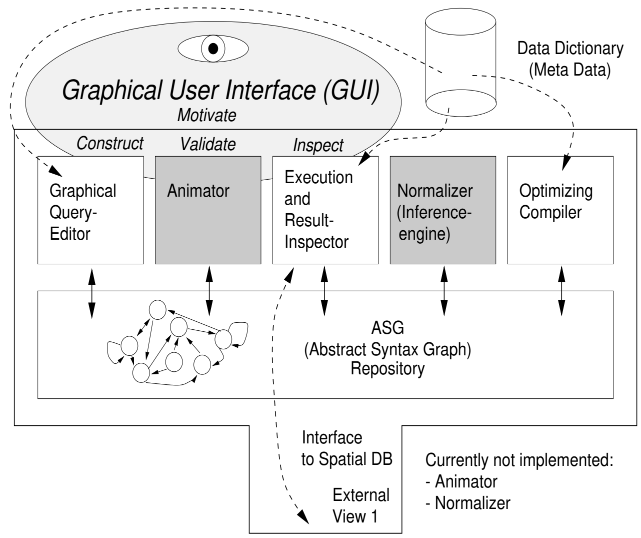

Figure 6: Logical architecture of VISCO.

visible inside of an enclosure.

- **Contains** is established between a new translucent enclosure and any other object that is completely contained within this new enclosure.

Figure 5 shows an example, where we search for three perfectly rectangular houses (of varying size) almost aligned in parallel that nearly lie on a straight line (please note that equivalent natural language descriptions are impossible). The "U"s denote universal objects (as discussed above) that must not be present in the database. Therefore, the aggregate (or transparency film) composed of the three rectangular polygons is not explicitly available in the database, but must be built by the system. The same condition holds for the derived center points of the rectangles. This is in contrast to the rectangles which must be explicitly found in the database.

When editing a query it is important to distinguish explicit from implicit (emergent) polygons and polylines. Please note that a polygon is by itself an implicit object – for instance, in the case of a triangle we only have visual indicators for the sides of the triangle, but not for the triangle *itself*. As a solution to this problem, we introduce for each explicit polygon (such as the buildings in Figure 5) and polyline an iconic sign in form of a "picture of the picture" (reduced to a small size).[1] According to the above mentioned rules, the buildings have to be *disjoint* from one another and have to stay *inside* of the big constant surrounding enclosure. The (derived) centroids of the buildings have to stay *inside* of the smaller constant enclosure, so that the buildings will be almost aligned on a straight line.

3 The VISCO Prototype

The logical architecture of the VISCO prototype is shown in Figure 6 and could be termed as a "repository model" (see [4]). We think of VISCO as a component in the application layer of a spatial database working on an external data model (view) provided especially for VISCO. Of course, meta data plays a crucial role for answering questions like "What types of objects are present in the database?", etc. The GUI is given through the following two components:

- a (syntax directed) query editor (a specialized graphical editor)
- an execution control and query result inspector (including the map viewer, see below)

Another component is the optimizing compiler (see Section 4). The animator (intended to visualize parts of the extension of the current query through animations) and the query normalizer are not yet implemented.[2] These five components work on a common abstract syntactic representation of the current query, which is maintained and managed by the abstract syntax graph repository module (ASG module). The ASG is given in form of a directed multi-hypergraph. The syntax directed query editor enforces the construction of correct ASGs in this repository. Some meta data must also be reflected at the user interface — for instance, it does not make sense to allow the user to query a

[1] This idea is due to our colleague Ralf Möller.

[2] The inference engine could detect unsatisfiable queries before actually doing the search as well as derive additional constraints that were only implicitly implied in the query, therefore making them accessible to the optimizing compiler.

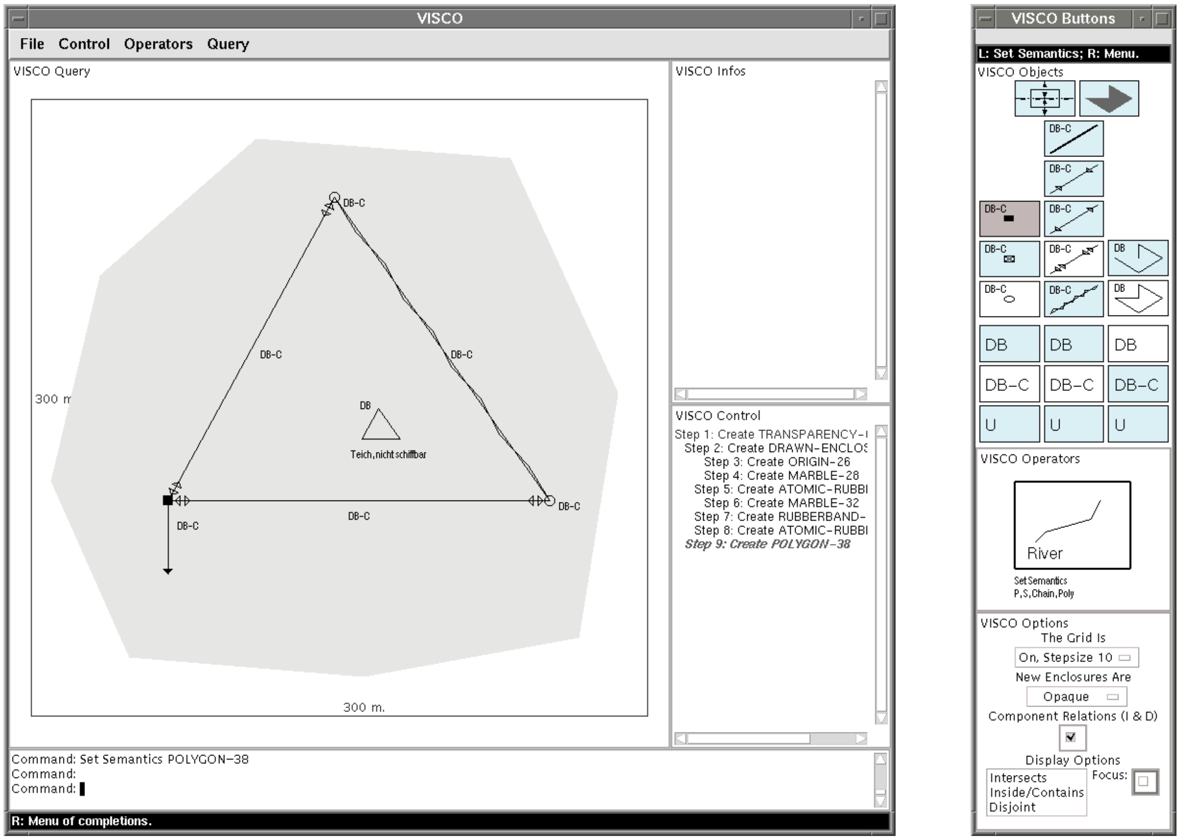

Figure 7: The Graphical Query Editor of VISCO: the main window (left) and the "Buttons" window (right)

CAD-database for buildings located in the vicinity of lakes. However, further investigations at the meta level must be done (e.g. see the work described in [6]).

We already emphasized the importance of well-designed and easy-to-use GUIs for visual languages. In our case, the graphical query editor shown in Figure 7 is one of the most important parts of the VISCO GUI. The editor's user interface is composed of two main windows, labeled "VISCO" and "VISCO Buttons" (handled by two communicating concurrent processes). The "VISCO" window is the *working space*, allowing users to interactively construct (execute, load, etc.) graphical queries. It consists of four main areas: the biggest one is the "VISCO Query" pane showing the actual graphical query (which has been already discussed, see Figure 1), the "VISCO Infos" pane providing helpful information and explanations, the command line for entering textual commands, and the "VISCO Control" pane displaying the editable query construction history, which is automatically maintained by the system during the construction of a graphical query (see below).

The *current state* of the query editor is maintained and completely visualized by the "VISCO Buttons" window. For instance, by selecting the button for "rubberband" in the "VISCO Objects" pane, the next new line segment will be created as a rubberband. The buttons are named as follows (from top to bottom and left to right): transparency, constant enclosure, beam, $\leq$-antenna, origin, $\geq$-antenna, nail, antenna, polyline (chain), marble, rubberband, polygon). The (slightly set apart) block of 9 buttons labeled "DB DB-C U" determines whether the next new geometric VISCO object will be a *geometric universal object* ("U"), or a *geometric query object* ("DB-C" or "DB" – the difference between these two can be ignored here). The two blocks of buttons in this pane must be considered columnwise; please note that the object icons of the buttons are labeled according to the actual selection (here, "DB-C", "DB-C", "DB").

Some other graphical presentation options can be selected with buttons in the "VISCO Options" pane, as well as an operator from an *iconic operator library* that pops up by pushing the currently active operator icon shown in the "VISCO Operators" pane. Then, the selected operator can be applied to a VISCO object (or a pair of VISCO objects in the case of a binary operator) visible in the "VISCO Query" area (prefix operator mode). Most commands can either be entered textually via the command line or chosen from the "Operators" menu in the menu line, as well as directly acti-

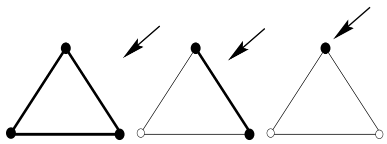

Figure 8: Vague gestures in VISCO.

vated on an object by a key sequence. However, most of the construction (e.g. polygon "drawing" etc.) is done directly without the need to refer to these operators.

In summary, the query editor offers the following powerful features:

Parallel maintenance of an *editable construction history*: By selecting a construction step in the history, the main display is updated to reflect the state of the query construction at the time of this step. Entries can be deleted (but of course, the "delete" operation can also be applied directly to their graphical counter parts). The indentation of entries shown in the "VISCO Control" pane reflects the graph structure of the objects (e.g., lines having end points as parts).

Facilities that support the users's process of query understanding and formulation: For instance, the recognized topological relationships (see above) for a newly introduced object can be visualized by coloring the already visible objects, denoting the enforced corresponding spatial constraint (blue means disjoint, red means intersects, green means inside/contains). By changing the *focus* or *current object* (this can be done in the "VISCO Control" pane), every enforced spatial constraint for every object can be inspected in a stepwise manner.

Handling of and interaction with complex objects: For instance, in the case of a polygon, the polygon as a whole as well as its segments and their end points must be referenced and manipulated by mouse gestures through the user. VISCO offers two mechanisms for achieving this goal: *first,* the notion of a *focus* (or *current*) *object* (which can be selected by pointing at the graphical object or its counterpart in the construction history); and *second* the concept of *vague mouse gestures* (see Figure 8). The current selection (displayed in bold) depends on the distance between the mouse pointer and the possibly targeted objects and their size.

"Top Down" and "Bottom Up" creation of complex objects: For instance, in order to create a polygon, a user must be able to select already present segments that afterwards become components of the freshly built polygon (what we call bottom up creation or aggregation), as well as to create some completely new segments and their end points (what we call top down creation). If a new object happens to be created, its type (e.g., rubberband, beam, antenna, etc.) and other attributes (should a new enclosure be opaque or translucent?) will be determined by the current state or mode of the query editor, which is maintained and visualized in the "VISCO Buttons" window. The editor's state can be changed concurrently *while* performing a complex operation (e.g. *while* creating a polygon, one segment could be defined as a rubberband between marbles, but the next segment could be a beam between two nails).

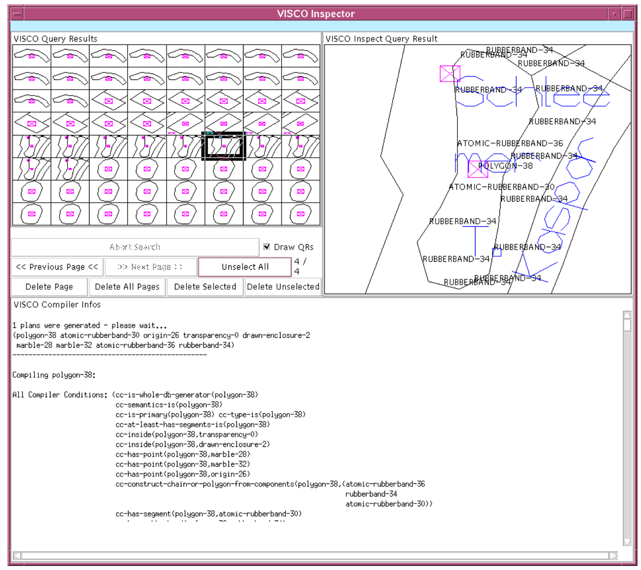

Figure 9: Execution and result inspection

Handling of and interaction with emergent objects: For instance, there has to be a way to materialize the emergent rectangle formed by two overlapping rectangular polygons or the intersection point formed by two intersecting lines. In this example, the emergent rectangle can easily be made explicit by a twofold application of the operator "Create intersection point" and an aggregation of the four points to create a new polygon. This is also an example for the use of *derived or computed objects*: after repositioning one of the intersecting lines, an automatic reconstruction of the scene must follow.

Other features of VISCO, such as unrestricted multi-level "Undo" and "Redo" (in our query editor, even a "Load" can be undone), context-sensitive help facilities etc. are considered as *obligatory* nowadays.

Figure 9 shows the query execution and result inspection component, another important part of the VISCO GUI. Here, the result of the query displayed in Figure 7 is shown. Each tile represents a match (in this case, a lake). Another pane shows the LISP code generated by the compiler for performing the search. Because components of polygons etc. are considered as individual objects, also permutations of "one and the same constellation" appear. Single tiles can be selected and further inspected, deleted, etc. Once a tile is selected, it can also be inspected more thoroughly with our advanced map inspection tool "Map Viewer" that is shown

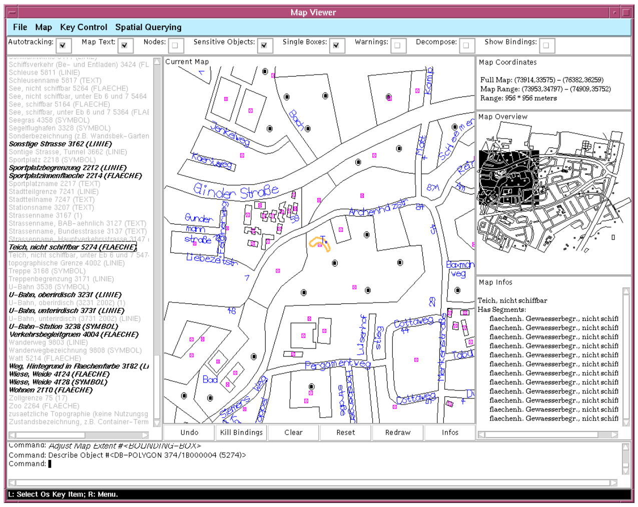

Figure 10: The Map Viewer

in Figure 10: here, also the neighborhood of a match can be inspected, neighborhood objects can be queried for their type by selecting them, their structure, etc. The map viewer also supports the generation of *layers* by selecting individual themes (each theme can be switched "on" or "off" in the scrollable list at the left side).

4 Representing and Compiling Queries

The ASG repository maintains an abstract syntactic representation of the current query in form of a directed multi-hypergraph. The nodes represent objects (e.g. marbles, rubberbands, etc.) with their properties (e.g. an object is a "lake"), the simple edges denote spatial relations and other constraints (e.g. direct component of). Hyperedges represent binary (ternary, ...) operators (e.g. a marble can be the derived intersection point of two beams). The ASG is constructed by a sequence of internal operator applications provided by the ASG module, each checking its applicability by a list of *preconditions*. By the enforcement of these preconditions we ensure that only syntactically correct ASGs can be constructed. Most of the user's interactions can directly be mapped to sequences of these internal operators. No advanced parsing techniques from visual language theory are necessary; however, the above mentioned topological relationships between language elements are recognized by algorithms borrowed from computational geometry.

In fact, the graphical query editor maintains an internal construction or application sequence of these internal ASG operators (a beautified subset of this sequence is shown in the "VISCO Control" window). After a user operation, the internal history is updated: an entry can be added, deleted or modified. In the case of the removal or modification of an entry of the history, the ASG is simply reconstructed by

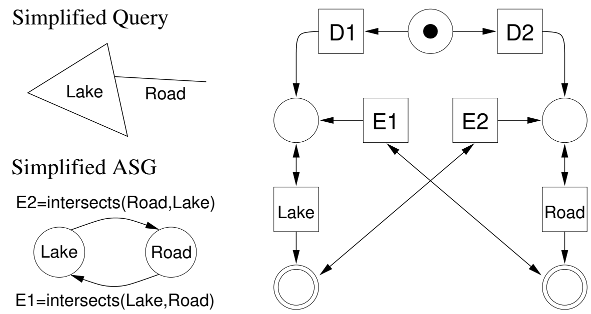

Figure 11: A simple ASG and its corresponding C/E net

replaying the whole internal history. If an error is encountered during the reconstruction phase because of an unfulfilled precondition, the user's interaction is regarded as invalid and automatically undone by the system. However, in our experience this "frustrating situation" does not appear too often (the only critical interactions regarding this are "move" and "delete"). The reconstruction of the ASG is fast enough and therefore appears instantaneously to the user.

The process of *query compilation* can be easily explained by using a *petri net model* (in fact, the compiler can be viewed as a special petri net). A VISCO query has to be considered as highly declarative — many possible *execution plans* can be expected. The *optimizer* determines the best of these plans (by assigning to plans cost weights) and uses this plan to construct a LISP program that will search the database simply in a depth-first manner (backtracking with "generate-and-test"). A plan itself is basically a sequence of nodes of the ASG representing the order of sequence in which query objects are matched to objects in the spatial database. However, to find the best of these potentially $n!$ number of plans is a very hard problem and can only be addressed heuristically.

In contrast to query objects, universal objects must be constructed or calculated and can not be searched for (either their operands have to be known and bound in the case of derived universal objects or their component objects in the case of complex universal objects). This demonstrates the possibility of having a large number of dependencies between objects that might reduce the number of possible execution (search) plans.

In Figure 11, a very simple ASG and its corresponding C/E petri net is shown.[3] The nodes "Lake" and "Road" (representing query objects) stand in an "intersects" relation (note that by taking the component relations for the – here not shown – endpoints of the road into account, we

[3]In a *condition event (C/E) net*, each *place* has a maximal capacity of one – a *transition* is *activated*, if each of its in-places is marked with *tokens* and all its out-places are empty *after* removing all tokens from the in-places.

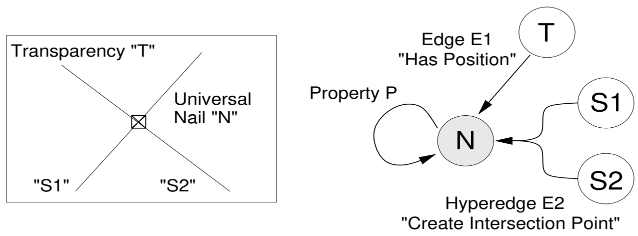

Figure 12: A node "N" of a more complex ASG

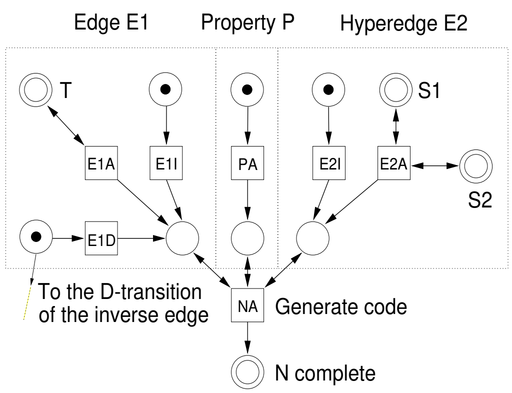

Figure 13: Corresponding C/E net for node "N"

would really get something like "touching"). Two plans are possible: first, we could search for "Lake" and then for "Road" by using the edge E2 as a *generator* (we use a *spatial index* supporting spatial join and selection operations based on topological relationships). This requires that the edge E1 has to be *deferred.* But the reverse order is also possible. Multiple plans can be derived by traversing edges or their inverses. The *possible plans* are now given as *processes* (sequences of firing transitions) of this simple net (the transitions D1 and D2 stand for "defer E1" and "defer E2"): D1–Lake–E2–Road or D2–Road–E1–Lake.[4] A firing transition usually generates LISP code for the search program. In a *complete* plan, each double-circled place must be marked. The quality of the generated plans can differ dramatically (factor 10.000 or more). The heuristics used by the optimizer can not be discussed here.

A more complex example is shown in Figure 12 and Figure 13. Sometimes, additional complex dependencies that can not be modeled by simple C/E nets must be taken into account. Therefore, transitions might be annotated with additional predicates that must be fulfilled before firing (in addition to the firing rule). Edges might be *ignored* under certain circumstances if their corresponding condition is already *implied* by the pre-history (see the annotated transitions E1I, E2I).

[4] The first plan is better since we have many roads intersecting other elements (e.g. roads), but only very few lakes.

5 Conclusion and Future Work

We presented an advanced prototype (fully implemented and operational) for an innovative new sketch-based visual spatial query language. A major advantage of our approach is the direct visibility of an objects meaning. The strong physical metaphors for language elements make the intended semantics and therefore the interpretation chosen by the system explicit. The built-in browser described here facilitates a step-wise focusing and understanding of the current query as a whole. Mismatches between the users intended mental meaning of the query and its interpretation as computed by the system are therefore mostly avoided. We argue that it can be very difficult or even impossible (at the current stage of art in artificial intelligence) to correctly grasp the users intentions (the relevant aspects) from a *freestyle drawn sketch* (like the ones assumed in [7]). Therefore, in order to get a practical system working *today*, the gap can not be bridged by the system alone. We still need the semantic input from users and we need systems offering active support, so that both are meeting halfway. The described GUI provides an embedding spatial querying environment, regarding querying as an incremental, step-wise process of selections, result inspections and further refinements. However, the system is still in its very early days and has not been extensively tested by users or been evaluated yet. Due to lack of space, we refer to [1] for a more complete discussion of related work.

References

[1] V. Haarslev and M. Wessel, "Querying GIS with animated spatial sketches", in *1997 IEEE Symposium on Visual Languages, Capri, Italy, Sep. 23-26*. Sept. 1997, pp. 197–204, IEEE Computer Society Press, Los Alamitos.

[2] T. Catarci, M.F. Costabile, S. Levialdi, and C. Batini, "Visual query systems for databases: A survey", *Journal of Visual Languages and Computing*, vol. 8, no. 2, pp. 215–260, Apr. 1997.

[3] M. Graf, "Visual Programming and Visual Languages: Lessons Learned in the Trenches", in *1990 IEEE Workshop on Visual Languages*. 1990, IEEE Computer Society Press.

[4] I. Sommerville, *Software Engineering*, Addison-Wesley, 5. edition, 1995.

[5] M.J. Egenhofer, "Why not SQL!", *International Journal on Geographical Information Systems*, vol. 6, no. 2, pp. 71–85, 1992.

[6] V. Haarslev, "A fully formalized theory for describing visual notations", in *Visual Language Theory*, K. Marriott and B. Meyer, Eds. Springer Verlag, Berlin, 1998, In press.

[7] M.J. Egenhofer, "Spatial-query-by-sketch", in *1996 IEEE Symposium on Visual Languages, Boulder, Colorado, USA, Sep. 3-6*. Sept. 1996, pp. 60–67, IEEE Computer Society Press, Los Alamitos.