Published in: Proceedings, 12th IEEE Symposium on Visual Languages, Boulder, Colorado, USA, Sep. 3-6, 1996, IEEE Press, 1996, in press.

GenEd – An Editor with Generic Semantics for Formal Reasoning about Visual Notations

Volker Haarslev and Michael Wessel
University of Hamburg, Computer Science Department,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
`http://kogs-www.informatik.uni-hamburg.de/~haarslev/`

Abstract

We describe the object-oriented editor GenEd supporting the design of specifications for visual notations. Prominent features of GenEd are (1) it is generic, *i.e. domain-specific syntax and semantics are specified by users; (2)* built-in parser *for actual drawings, driven by formal specifications; (3) powerful* reasoning *capabilities about diagrams and their specification. GenEd's specification language is based on a fully formalized theory for describing visual notations. Three examples, place-transition petri nets, entity-relationship diagrams, and a small GIS application are presented.*

Keywords— Theory of visual languages, formal semantics, diagrammatical reasoning, description logics, visual editor, visual parsing, geographical information systems.

1 Motivation and Introduction

This paper reports on the design and evaluation of the generic editor GenEd for visual notations. The term "generic" refers to the characteristic that users can specify domain-specific syntax and semantics for visual notations. We think of formal notations as a very general notion. Examples for visual notations can be found in areas such as music, geography, mathematics, chemistry, etc. Visual programming languages are viewed as a subset of formal notations.

GenEd is intended for supporting the design and analysis of visual notations or visual programming languages. Usually the design of new visual notations is an exploratory approach. Our experience with the design of formal semantics [1] for a completely visual programming language, Pictorial Janus [2, 3], has strongly motivated the approach presented in this paper. We argue that designers of notations should be offered an almost free-form, purely declarative style for specifications with immediate support for analysis and reasoning about specifications and for verification through parsing of example drawings. GenEd's parser can operate in two modes. The *incremental* mode validates drawings after every relevant modification and reports parsing successes and errors to the user. The second mode performs parsing only when demanded by the user.

Our approach is based on a fully formalized theory about visual notations that can be defined from "first principles" by starting with point-sets and topology. GenEd's specification language and its reasoning capabilities depend on description logic theory.

This paper makes several contributions to visual languages. It demonstrates that our theory can be applied to a variety of different visual notations, ranging from simple diagrams, over petri nets, to sophisticated visual programming languages. The latter was extensively described in [1]. It also introduces a new full-featured graphic editor based on our formal theory. This theory was introduced in [1] and is extensively presented and related to visual language theory in [4]. Our approach is in contrast to other approaches that use only syntax specifications or favor a generative solution, i.e. they create specialized editors for particular visual languages.

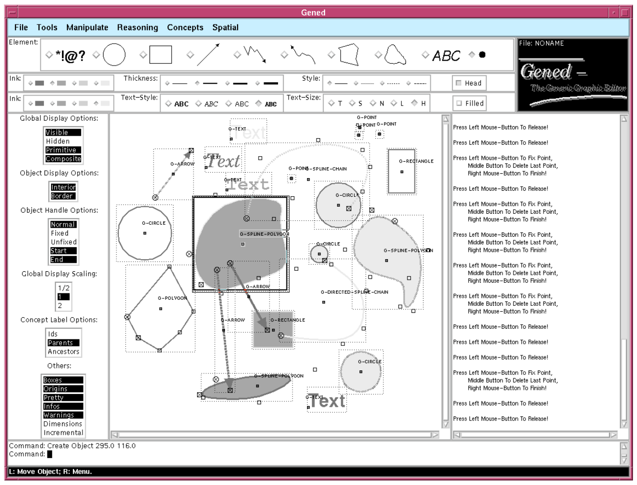
The next section describes our theoretical foundation in more detail. Then follows the main section explaining GenEd's user interface and presenting three example applications. The last sections discuss related work and future research.

2 Theoretical Foundation

We believe that the semantics of formalisms used for VL theory should be well understood. That is, the meanings of represented language concepts should be unambiguously determined by explicit notational formalisms, so that algorithms can operate on the representation in accordance with the semantics of the notation, without needing ad hoc provisions for specific VL domains. Our approach is based on a fully formalized theory [4] for describing visual notations that consists of three components. Each component is defined by precise semantics. Objects and relations are defined by point-sets and topology. Description logic theory can be based on model-theoretic semantics using a compositional axiomatization with set theory.

2.1 Geometrical Objects

GenEd implements these three components in accordance to our theory. However, the implementation of geometrical objects and their corresponding spatial re-



```
Press Left Mouse-Button To Release!

Press Left Mouse-Button To Release!

Press Left Mouse-Button To Fix Point,
      Middle Button To Delete Last Point,
      Right Mouse-Button To Finish!

Press Left Mouse-Button To Release!

Press Left Mouse-Button To Fix Point,
      Middle Button To Delete Last Point,
      Right Mouse-Button To Finish!

Press Left Mouse-Button To Fix Point,
      Middle Button To Delete Last Point,
      Right Mouse-Button To Finish!

Press Left Mouse-Button To Fix Point,
      Middle Button To Delete Last Point,
      Right Mouse-Button To Finish!

Press Left Mouse-Button To Fix Point,
      Middle Button To Delete Last Point,
      Right Mouse-Button To Finish!

Press Left Mouse-Button To Release!

Press Left Mouse-Button To Release!

Press Left Mouse-Button To Release!

Press Left Mouse-Button To Fix Point,
      Middle Button To Delete Last Point,
      Right Mouse-Button To Finish!

Press Left Mouse-Button To Release!

Press Left Mouse-Button To Release!
```

```
Command: Create Object 295.0 116.0
Command:
```

Figure 1: Examples for geometric objects.

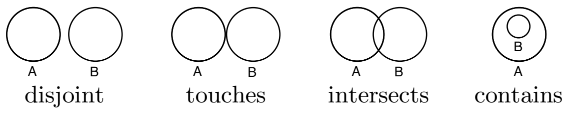


Figure 2: Primitive relations between A and B

lations uses well-known computer graphics techniques for reasons of efficiency. The semantics of these algorithms are still specified within our theory (see [5] for a complete treatment). GenEd offers a set of predefined geometrical objects (similar to other object-oriented graphic editors) that can be used to design examples of particular notations. Supported primitive objects are points, (directed) line segments, line segment chains, and (spline) polygons (see Figure 1).

2.2 Spatial Relations

GenEd recognizes seven primitive spatial relations (*disjoint, touches, intersects, contains/contained_by, covers/covered_by*) that may hold between two objects (see Figure 2). It also computes the dimension of the intersection, if applicable. The semantics are defined in analogy to a proposal by Clementini et al. [6] and are based on point-sets and topology. The relations have a parameterized 'fuzziness' compensating for inexact positioning of objects (caused by users or scaling factors) and floating-point arithmetic. In contrast to several other approaches for spatial relations (e.g. see [4]) GenEd can also deal with concave objects. Additionally, an arbitrary collection of objects may be grouped together and treated as a *composition* object. Analogous semantics for composition objects were defined.

Higher-level relations can be defined with the help of the above mentioned seven relations (see Figure

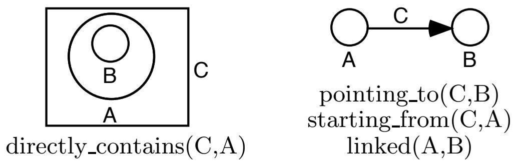


Figure 3: Higher-level relations

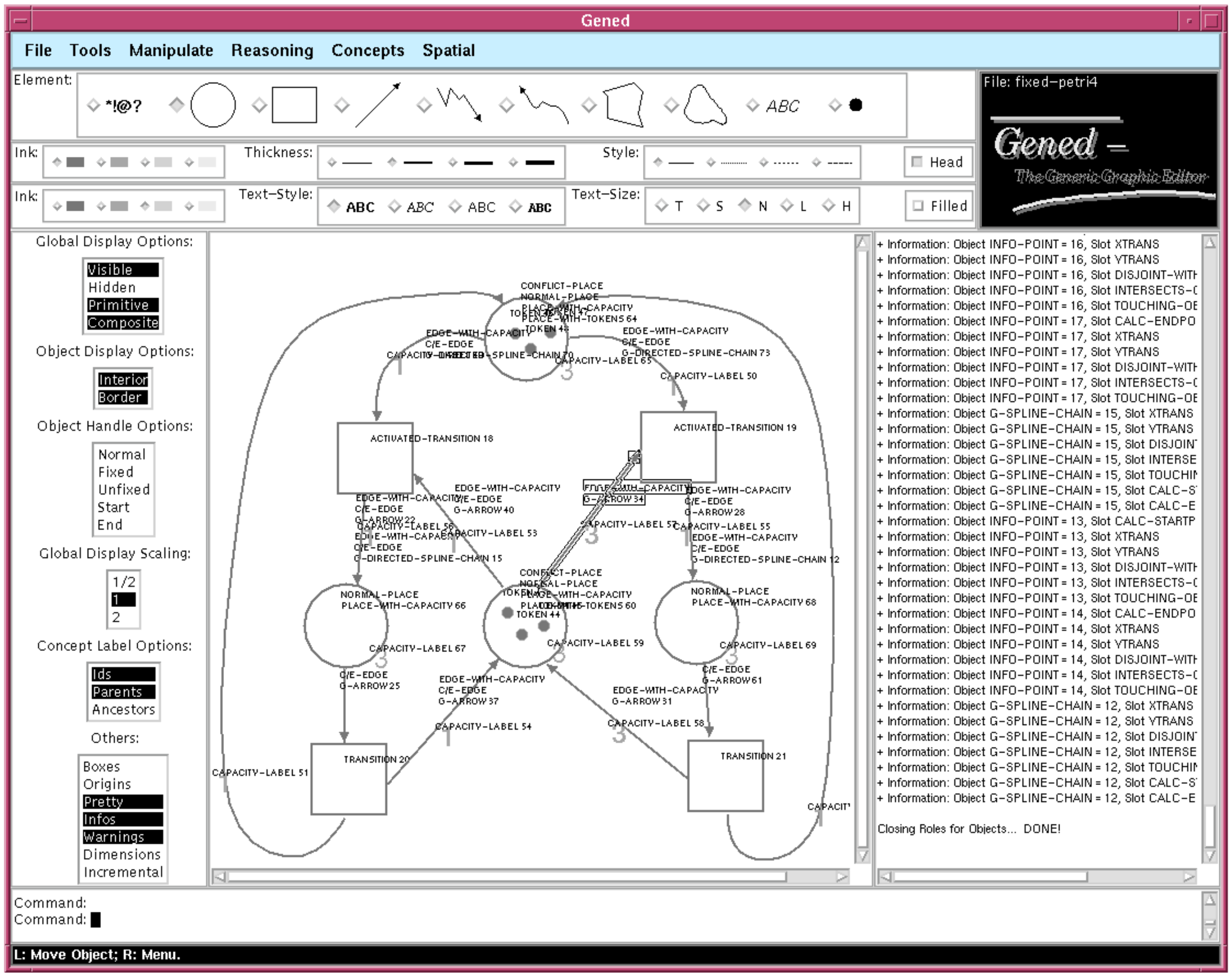
3). GenEd currently recognizes specialized containment (directly-contains/inside), connectivity (linked-with), and direction of line segments (starting-from, pointing-to). These relations are also applicable to composition objects.

2.3 Description logic

Description logic (DL) theory has been successfully applied to the specification of Pictorial Janus (PJ) [2, 3], a completely visual language for concurrent logic programming [1, 7].

DL theories are based on the ideas of structured inheritance networks [8]. A DL can be considered as a term rewriting language restricting the left side of equations to single unique term names. The specification of a DL consists of a set of *concepts* (or terms), a set of *roles* (binary relations that may hold between individuals of concepts), a set of disjointness assertions among concepts and among roles, a set of concept membership assertions for individuals, and a terminology, which maps names to specifications of concepts or roles. Concepts may be *primitive* or *defined*. A specification of a primitive concept represents conditions that are necessary but *not* sufficient. The specification of a defined concept represents conditions that are both necessary *and* sufficient. Primitive and defined roles are similarly specified. If a role holds between individuals, these individuals are referred to as *fillers* of this role. The meaning of DL theory can be described by model-theoretic semantics using a compositional axiomatization with set theory. The appendix summarizes the semantics of DL language elements that are used in the following sections.

There exist several theorem provers for special types of DLs. These theorem provers (referred to as DL systems) offer powerful reasoning mechanisms that are based on the DL semantics. DL systems usually distinguish two separate reasoning components. The *terminological reasoner* or *classifier* (TBox) classifies concepts with respect to subsumption relationships between these concepts and organizes them into a taxonomy. The TBox language is designed to facilitate the construction of concept expressions describing classes (types) of individuals. The classifier automatically performs consistency checking (e.g. for incoherence, cy-



```
+ Information: Object INFO–POINT = 16, Slot XTRANS
+ Information: Object INFO–POINT = 16, Slot YTRANS
+ Information: Object INFO–POINT = 16, Slot DISJOINT–WITH
+ Information: Object INFO–POINT = 16, Slot INTERSECTS–(
+ Information: Object INFO–POINT = 16, Slot TOUCHING–OE
+ Information: Object INFO–POINT = 17, Slot CALC–ENDPO
+ Information: Object INFO–POINT = 17, Slot XTRANS
+ Information: Object INFO–POINT = 17, Slot YTRANS
+ Information: Object INFO–POINT = 17, Slot DISJOINT–WITH
+ Information: Object INFO–POINT = 17, Slot INTERSECTS–(
+ Information: Object INFO–POINT = 17, Slot TOUCHING–OE
+ Information: Object G–SPLINE–CHAIN = 15, Slot XTRANS
+ Information: Object G–SPLINE–CHAIN = 15, Slot YTRANS
+ Information: Object G–SPLINE–CHAIN = 15, Slot DISJOIN
+ Information: Object G–SPLINE–CHAIN = 15, Slot INTERSE
+ Information: Object G–SPLINE–CHAIN = 15, Slot TOUCHIN
+ Information: Object G–SPLINE–CHAIN = 15, Slot CALC–S
+ Information: Object G–SPLINE–CHAIN = 15, Slot CALC–E
+ Information: Object INFO–POINT = 13, Slot CALC–STARTP
+ Information: Object INFO–POINT = 13, Slot XTRANS
+ Information: Object INFO–POINT = 13, Slot YTRANS
+ Information: Object INFO–POINT = 13, Slot DISJOINT–WITH
+ Information: Object INFO–POINT = 13, Slot INTERSECTS–(
+ Information: Object INFO–POINT = 13, Slot TOUCHING–OE
+ Information: Object INFO–POINT = 14, Slot CALC–ENDPO
+ Information: Object INFO–POINT = 14, Slot XTRANS
+ Information: Object INFO–POINT = 14, Slot YTRANS
+ Information: Object INFO–POINT = 14, Slot DISJOINT–WITH
+ Information: Object INFO–POINT = 14, Slot INTERSECTS–(
+ Information: Object INFO–POINT = 14, Slot TOUCHING–OE
+ Information: Object G–SPLINE–CHAIN = 12, Slot XTRANS
+ Information: Object G–SPLINE–CHAIN = 12, Slot YTRANS
+ Information: Object G–SPLINE–CHAIN = 12, Slot DISJOIN
+ Information: Object G–SPLINE–CHAIN = 12, Slot INTERSE
+ Information: Object G–SPLINE–CHAIN = 12, Slot TOUCHIN
+ Information: Object G–SPLINE–CHAIN = 12, Slot CALC–S
+ Information: Object G–SPLINE–CHAIN = 12, Slot CALC–E

Closing Roles for Objects... DONE!
```

```
Command:
Command:
```

Figure 4: GenEd: petri net for reader-writer problem (simplified)

cles) of concept definitions and offers retrieval facilities about the classification hierarchy. The forward-chaining *assertional reasoner* or *realizer* (ABox) recognizes and maintains the type (concept membership) of individuals. The purpose of the ABox language is to state constraints or facts (usually restricted to unary or binary predications) that apply to a particular domain or world. Assertional reasoners support a query language as access to their state. Some query languages offer the expressiveness of the full first-order calculus.

GenEd defines several predefined primitive concepts resembling supported geometric objects. It also introduces a set of primitive roles representing the spatial relations described in the previous section. The built-in spatial parser of GenEd can recognize these objects and their spatial relationships. GenEd can create for elements of a drawing a corresponding set of ABox elements asserting concept membership (e.g. geometric objects A is a rectangle) and role fillers (e.g. rectangle A touches line B). GenEd's roles are organized in a hierarchy with inheritance for role fillers. Roles with at most one filler are referred to as *attributes* (e.g. text_value for text elements). Many geometric attributes are available for objects (size, position, ink, etc). GenEd also maintains part-of relationships for composition objects.

3 GenEd

Figure 4 shows the user interface in detail. The workspace window displays a fully classified petri net example (see Section 3.3.1 for explanations). The next sections describe GenEd's user interface, its implementation, and two example applications.

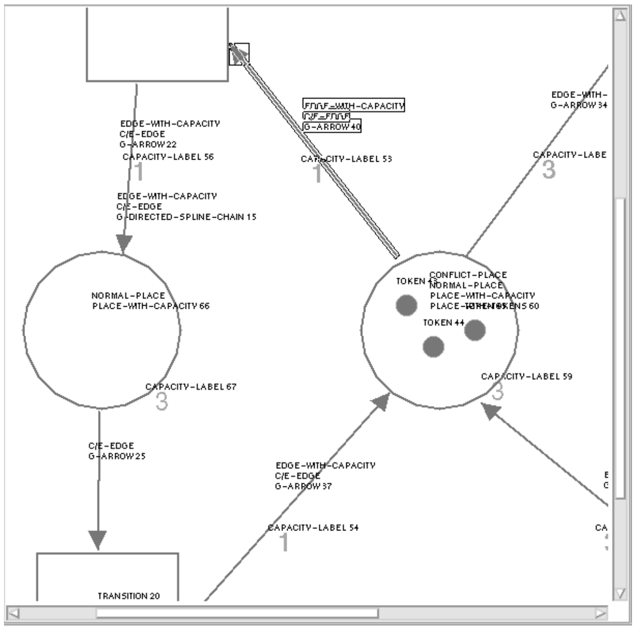


Figure 5: GenEd: magnified selection of petri net

3.1 User Interface

The user interface contains several (scrollable) panes and a menu bar at the top. The three horizontal panes below the menu bar offer the selection of object types and the setting of drawing attributes for elements and text. The left vertical pane shows a variety of modifiable parameters for controlling display options and scaling factors. The center pane (workspace) contains a petri net for the reader-writer problem. It displays the petri net elements (place as circle, transition as rectangle, edge as (spline) arrow, token as bullet, capacity label as gray number). The elements are also labeled with the concept names as computed by the classification phase of the spatial parser. The right pane is used to inform the user about computed concept memberships, role fillers, etc. The horizontal pane below the three vertical panes is the command pane. Users have the choice whether they enter commands as gestures (mouse movement, clicks) or as text commands. The pane at the bottom always shows object-sensitive documentation about available gestures.

Users can always select a collection of elements in the workspace with an enclosing bounding box and aggregate them into a composition object. The contents of the workspace can be zoomed in or out (see Figure 5 for a magnified selection of the petri net). In general, GenEd offers many operations on objects that are also available in commercial graphic editors (create, delete, copy, move, scale, rotate, hide, show, inspect, arrangement, save, restore, undo list).

It is worth to note that special handles might be attached to arbitrary objects. These handles can be used to fix relative positions between objects or to define stretchable lines whose end points might be fixed at objects. Primitive and composition objects may be stored in and retrieved from a user-defined library. Figure 6 shows a submenu displaying visualizations of petri net places stored in the library. The workspace can be saved in and loaded from a file.

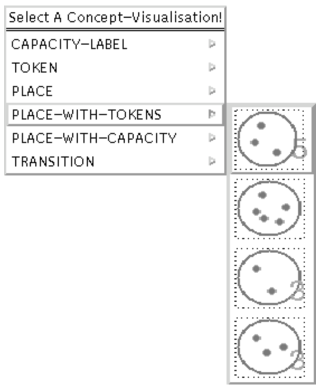


Figure 6: Library menu with submenu for places.

3.2 Implementation

GenEd is implemented in Common Lisp using the Common Lisp Object System (CLOS) and the Common Lisp Interface Manager (CLIM) as interface toolkit. The classification of concepts and the parsing of actual drawings take place by using CLASSIC [9, 10] as DL system. CLASSIC is also implemented in Common Lisp. GenEd consists of 28 modules with a total of about 300 KB source code (without CLIM, CLOS, and CLASSIC). GenEd is fully implemented with the features described in this paper.

3.3 Example Session

We demonstrate two visual notations whose specifications were created with GenEd. The first notation describes place-transition petri nets. The second notation defines entity-relationship (ER) diagrams.

3.3.1 Petri Nets

A *petri net* is a triple $N = (P, T, E)$ with P a set of places, a set $T \neq \emptyset$ of transitions, and a relation $E \subset (P \times T) \cup (T \times P)$ representing edges.

A tuple $N = (P, T, E, C, W, M)$ defines a *place-transition net* if the following conditions hold. The tuple (P, T, E) is a petri net with places P and transitions T. $C : P \rightarrow \mathbb{N} \cup \{\omega\}$ defines a capacity for each place. $W : E \rightarrow \mathbb{N} - \{0\}$ specifies the weight of every edge. $M : P \rightarrow \mathbb{N} \cup \{\omega\}$, with $\forall p \in P : M(p) \leq C(p)$, defines the initial marking.

The complete specification of place-transition nets is out of scope of this paper. We already proved in [1] with Pictorial Janus that our approach can handle complex visual notations. We only outline the design of the specification for place-transition nets. We define concepts representing legal constellations for places, transitions, and edges. A petri net is specified as a composite object consisting of at least 5 parts. Primitive concepts are typeset in a slanted style.

petri_net $\equiv$
(*composite_thing* $\wedge$
($\exists_{\geq 5}$ has_parts) $\wedge$ ($\exists_{\geq 1}$ has_parts place) $\wedge$
($\forall$ (has_parts$\bullet$ *rectangle*) transition) $\wedge$
($\forall$ (has_parts$\bullet$ *arrow*) edge))

Petri nets are specialized to place-transitions nets after defining capacity labels, places with capacity, token, places with tokens, edges with capacity, and active transitions.

place_transition_net $\equiv$
(petri_net $\wedge$ ($\exists_{\geq 1}$ has_parts place_with_token))

An interesting special case of place-transition nets is a *predicate-event* net. All places and edges have a maximal capacity of 1.

$$\forall p \in P\colon C(p) = 1 \wedge \forall (x,y) \in E : W(x,y) = 1$$

predicate_event_net $\equiv$
(place_transition_net $\wedge$
($\forall$ (has_parts$\bullet$ place) predicate_event_place) $\wedge$
($\forall$ (has_parts$\bullet$ *arrow*) predicate_event_edge))

The definition of predicate-event places and transitions are omitted. There are still other interesting concepts characterizing special petri net elements that are left out due to lack of space. However, the next section discusses entity-relationship diagrams which are specified in almost full detail.

3.3.2 Entity-Relationship Diagrams

Our definition of a subset of entity-relationship (ER) diagrams was inspired by [11]. We present this example in order to demonstrate the expressiveness of our specification language and the reasoning capabilities of GenEd. Due to space constraints we show only a screen shot of GenEd (see Figure 7) displaying a subpart of a larger example modelling relationships in an airline company. Figure 8 shows a simple ER diagram specifying a relationship between a pilot and an aircraft. The design and evaluation of the specification for this simplified type of ER diagrams took about half an hour for an experienced user of GenEd. We assume a few primitive concepts and spatial relations representing geometrical objects (rectangle, circle, diamond, line, text) used in our ER diagram language.

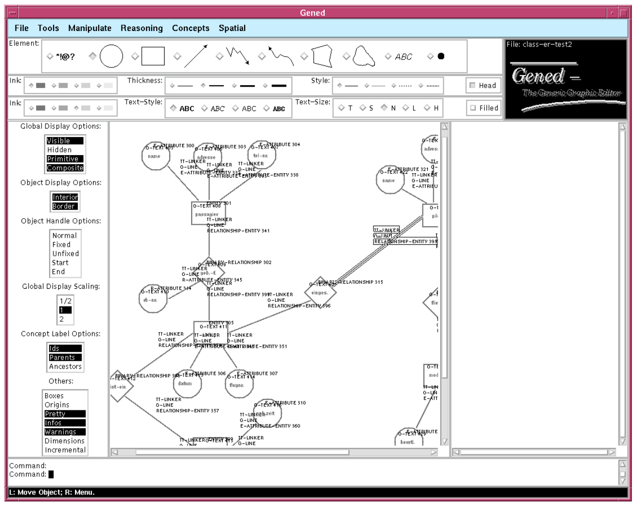


Figure 7: An ER diagram modelling airlines

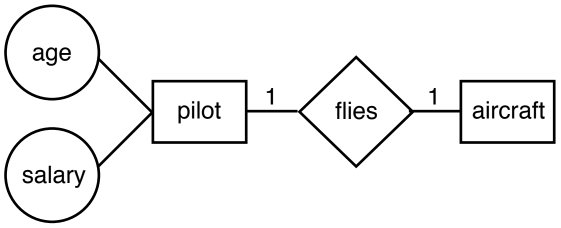
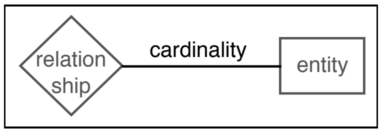


Figure 8: A simple entity-relationship diagram

Connectors

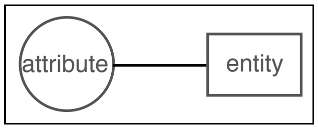


A *relationship-entity* connection is a line that touches exactly one text label (expressing cardinality) and two other regions (rectangle or diamond). A *cardinality* is a text string with values chosen from the set $\{1, m, n\}$.

relationship_entity $\equiv$
(*line* $\wedge$ ($\exists_{=3}$ touching) $\wedge$ ($\exists_{=1}$ touching *text*) $\wedge$
($\exists_{=2}$ touching (*rectangle* $\vee$ *diamond*)) $\wedge$
($\exists_{=1}$ touching *rectangle*) $\wedge$ ($\exists_{=1}$ touching *diamond*))

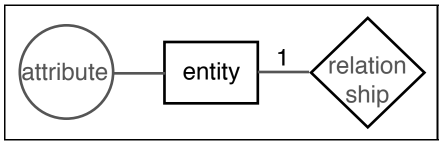
cardinality $\equiv$
(*text* $\wedge$ ($\forall$ touching relationship_entity) $\wedge$
($\exists_{=1}$ touching) $\wedge$ ($\forall$ text_value $\{1, \mathrm{m}, \mathrm{n}\}$))

An *attribute-entity* connection is a line that touches only two regions (circle or rectangle) and no text string.



attribute_entity $\equiv$
$(\mathit{line} \wedge (\exists_{=2}\ \mathsf{touching}) \wedge$
$(\forall\ \mathsf{touching}\ (\mathit{circle} \vee \mathit{rectangle})) \wedge$
$(\exists_{=1}\ \mathsf{touching}\ \mathit{rectangle}) \wedge (\exists_{=1}\ \mathsf{touching}\ \mathit{circle}))$

Entities



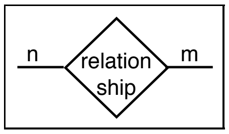
An *entity* is a rectangle that contains its name. It touches at least one relationship-entity and optionally some attribute-entity connectors. It is linked with at least one diamond.

named_region $\equiv$
$(\mathit{region} \wedge (\exists_{=1}\ \mathsf{containing}) \wedge (\forall\ \mathsf{containing}\ \mathit{text}))$

entity $\equiv$
$(\mathit{rectangle} \wedge \mathsf{named\_region} \wedge$
$(\exists_{\geq 1}\ \mathsf{touching}\ \mathsf{relationship\_entity}) \wedge$
$(\forall\ \mathsf{touching}\ (\mathsf{attribute\_entity} \vee \mathsf{relationship\_entity})) \wedge$
$(\exists_{\geq 1}\ \mathsf{linked\_with}\ \mathit{diamond}) \wedge$
$(\forall\ \mathsf{linked\_with}\ (\mathit{circle} \vee \mathit{diamond})))$

Relationships

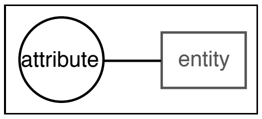
A *relationship* is a diamond that contains its name. It touches one relationship-entity and optionally some attribute-entity connectors. It is linked with two entities.



relationship $\equiv$
$(\mathit{diamond} \wedge \mathsf{named\_region} \wedge$
$(\exists_{=2}\ \mathsf{linked\_with}) \wedge (\forall\ \mathsf{linked\_with}\ \mathsf{entity}) \wedge$
$(\exists_{=2}\ \mathsf{touching}) \wedge (\forall\ \mathsf{touching}\ \mathsf{relationship\_entity}) \wedge$
$(\exists_{\leq 2}\ \mathsf{touching}\ (=\ (\mathsf{touching} \circ \mathsf{text\_value})\ 1)) \wedge$
$(\exists_{\leq 1}\ \mathsf{touching}\ (=\ (\mathsf{touching} \circ \mathsf{text\_value})\ \mathsf{m})) \wedge$
$(\exists_{\leq 1}\ \mathsf{touching}\ (=\ (\mathsf{touching} \circ \mathsf{text\_value})\ \mathsf{n})))$

Attributes

An *attribute* is a circle that contains its name. It touches one attribute-entity connector and is linked with an entity.



attribute $\equiv$
$(\mathit{circle} \wedge \mathsf{named\_region} \wedge$
$(\exists_{=1}\ \mathsf{linked\_with}) \wedge (\forall\ \mathsf{linked\_with}\ \mathsf{entity}))$

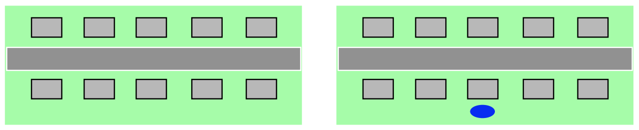


Figure 9: Sketches of the villa example.

3.3.3 Geographic Information Systems

The last example suggests that our approach might also be usable in the domain of geographic information systems (GIS). This example demonstrates the advantages of a forward-chaining parser and the necessity for non-monotonic reasoning. In the context of an aerial image interpretation system we assume that there exists a two-dimensional model of a geographic scene stored in a GIS. In this system multi-spectral images are interpreted, for example, in order to detect changes in the scene. To speak of an *interpretation* of an image, e.g. a change has to be described not only at the geometrical level but also at the conceptual level. We use GenEd to create a model of an example world. This small example consists of two primitive objects (water surface and house) that can be easily detected in aerial images by using spectral analysis. We assume a symmetric relation 'near' that may hold for objects if they are located next to each other.

A *swimming pool* is a water surface with at least one house in its neighborhood (relation 'near'). A *villa* is a house with at least one swimming pool in its neighborhood. The left part of Figure 9 sketches an image of a scene with several houses along a street. In this scene no villa has been recognized since there exists no water surface next to a house. It is important to note that each house could still incrementally be classified as villa provided a water surface is found in the neighborhood. This is visualized in the right part of Figure 9. This particular house is now classified as villa and the water surface as swimming pool. This reclassification is triggered by the newly added water surface and affects only houses in the neighborhood of this water surface. It is automatically performed by the forward-chaining parser and always results in the most-specialized classification (that is what we are interested in). A backward-chaining parser would be inadequate if we imagine a scene with a large amount of houses and only a small number of water surfaces since we usually do not know in advance where a water surface might be detected. Thus, it is also not acceptable to query the state of all houses. This scenario illustrates that GenEd might also be a suitable query interface for a GIS.

4 Related Work

Our work on GenEd is especially related to designing VL editors and to theory of VLs. There exist many approaches to the theory of visual language specifications and formalisms, but the number of system implementations is very limited. Mostly, these approaches extend string grammar formalisms. A complete and recent overview is out of scope of this paper. However, we like to mention a few approaches: generalizations of attributed grammars (e.g. picture layout grammars [12]), positional grammars (e.g. [13]), graph grammars (e.g. [14, 15, 16]), and algebraic or type-theoretic formalisms (e.g. [17, 18]). Other work closely related to our approach uses (constraint) logic or relational formalisms (e.g. [19, 20, 21, 22, 23, 24, 25]) for representing spatial relationships. A more detailed review of closely related work on VL theory can be found in [4].

GenEd's philosophy of a free-form general purpose editor supported by (incremental) visual parsing is in contrast to the following two approaches. Escalante [26] is an environment for the rapid construction of VL applications. It supports the construction of languages that are based on graph models (nodes and edges). The emphasis of Escalante is on user interface construction and not on VL semantics. [17] describes an approach to generate syntax-oriented visual editors for formally specified languages. It is embedded into an algebraic specification formalism but deals only with syntactic issues of VLs.

We conclude our review with two approaches that are closely related to GenEd. DiaGen [27] is a generator for diagram editors providing direct manipulation and animation of diagrams. Diagrammatic VLs can be specified by a hypergraph grammar which is used to generate a specialized editor. The extension of DiaGen by a parser supporting general editing is in progress. Another approach [28] is concerned with the automatic construction of (pen-based) user interfaces for visual languages specified by constraint multiset grammars. The VL specification is used to generate a language-specific parser. A general editor tool utilizes these parsers to control and support the editing process of corresponding VLs.

A major distinction to the approaches mentioned in this section is the focus of GenEd on the design of formal specifications of visual notations. Our experience suggests that this is an iterative process with mutual dependencies between the visual notation and its specification. GenEd supports this development cycle by offering a free-form graphic editor and object-centered specifications based on a formal theory with clear semantics. The general, built-in visual parser is directly driven by these specifications.

5 Future Research

We are currently working on specifications for other visual notations (e.g. venn diagrams) that represent mathematical notions. We are planning to incorporate concrete domains over the algebra of simple reals. These concrete domains extend description logic by reasoning about systems of (in)equalities over (non)linear polynomials (see [4] for more details). We also plan to address dynamic semantics of VLs. GenEd might be extended to support specifications of temporal relationships and to visualize VL execution through animations.

References

[1] V. Haarslev, "Formal Semantics of Visual Languages using Spatial Reasoning", In VL'95 [29], pp. 156–163.

[2] K.M. Kahn and V.A. Saraswat, "Complete Visualizations of Concurrent Programs and their Executions", in *1990 IEEE Workshop on Visual Languages, Skokie, Illinois, Oct. 4-6.* Oct. 1990, pp. 7–14, IEEE Computer Society Press.

[3] K.M. Kahn, V.A. Saraswat, and V. Haarslev, "Pictorial Janus: A Completely Visual Programming Language and its Environment (in German)", in *GI-Fachgespräch Programmieren multimedialer Anwendungen der GI-Jahrestagung 1991, Darmstadt, Oktober 1991*, J. Encarnacao, Ed. 1991, pp. 427–436, Springer Verlag, Berlin.

[4] V. Haarslev, "A Fully Formalized Theory for Describing Visual Notations", in *International Workshop on the Theory of Visual Languages, Gubbio, Italy*, May 1996.

[5] M. Wessel, "Development of a concept-oriented Generic Graphic Editor in Common Lisp (in German)", Jan. 1996, Studienarbeit.

[6] E. Clementini, P. Di Felice, and P. van Oosterom, "A Small Set of Formal Topological Relationships Suitable for End-User Interaction", in *Advances in Spatial Databases, Third International Symposium, SSD'93, Singapore, June 23-25, 1993*, D. Abel and B.C. Ooi, Eds. June 1993, vol. 692 of *Lecture Notes in Computer Science*, pp. 277–295, Springer Verlag, Berlin.

[7] V. Haarslev, "On Formal Semantics of Visual Notations", Technical Report, in preparation, 1996.

[8] R.J. Brachman and J.G. Schmolze, "An overview of the KL-ONE knowledge representation system", *Cognitive Science*, pp. 171–216, Aug. 1985.

[9] R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L.A. Reswnick, and A. Borgida, "Living with Classic: When and How to Use a KL-ONE-like Language", in *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, J.F. Sowa, Ed., San Mateo, California, 1991, pp. 401–456, Morgan Kaufmann Publishers.

[10] R.J. Brachman, "“Reducing” CLASSIC to Practice: Knowledge Representation Theory Meets Reality", in *Principles of Knowledge Representation and Reasoning, Third International Conference, Cambridge, Mass., Oct. 25-29, 1992*, Oct. 1992, pp. 247–258.

[11] J.A. Serrano, "The Use of Semantic Constraints on Diagram Editors", In VL'95 [29], pp. 211–216.

[12] E.J. Golin, "Parsing Visual Languages with Picture Layout Grammars", *Journal of Visual Languages and Computing*, vol. 2, no. 4, pp. 371–393, Dec. 1991.

[13] G. Costagliola, M. Tomita, and S.K. Chang, "A Generalized Parser for 2-D Languages", in *1991 IEEE Workshop on Visual Languages, Kobe, Japan, Oct. 8-11.* Oct. 1991, pp. 98–104, IEEE Computer Society Press.

[14] H. Göttler, "Graph Grammars, a new Paradigm for Implementing Visual Languages", in *Rewriting Techniques and Applications, 3rd International Conference, RTA-89, 3-5 April 1989, Chapel Hill, NC.* Apr. 1989, pp. 152–166, Springer Verlag, Berlin.

[15] M.A. Najork and S.M. Kaplan, "Specifying Visual Languages with Conditional Set Rewrite Systems", in *1993 IEEE Symposium on Visual Languages, Bergen, Norway, Aug. 24-27.* Aug. 1993, pp. 12–17, IEEE Computer Society Press.

[16] J. Rekers and A. Schürr, "A Graph Grammar Approach to Graphical Parsing", In VL'95 [29], pp. 195–202.

[17] S.M. Üsküdarli, "Generating Visual Language Editors for Formally Specified Languages", In VL'94 [30], pp. 278–285.

[18] D. Wang, J.R. Lee, and H. Zeevat, "Reasoning with Diagrammatic Representations", in *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, J. Glasgow, N.H. Narayanan, and B. Chandrasekaran, Eds., pp. 339–393. AAAI Press / The MIT Press, Menlo Park, California, 1995.

[19] C. Crimi, A. Guercio, G. Nota, G. Pacini, G. Tortora, and M. Tucci, "Relation Grammars and their Application to Multi-dimensional Languages", *Journal of Visual Languages and Computing*, vol. 2, no. 4, pp. 333–346, Dec. 1991.

[20] R. Helm and K. Marriott, "A Declarative Specification and Semantics for Visual Languages", *Journal of Visual Languages and Computing*, vol. 2, no. 4, pp. 311–331, Dec. 1991.

[21] B. Meyer, "Pictures Depicting Pictures: On the Specification of Visual Languages by Visual Grammars", in *1992 IEEE Workshop on Visual Languages, Seattle, Washington, Sept. 15-18.* Sept. 1992, pp. 41–47, IEEE Computer Society Press.

[22] K. Wittenburg, L. Weitzman, and J. Talley, "Unification-based Grammars and Tabular Parsing for Graphical Languages", *Journal of Visual Languages and Computing*, vol. 2, no. 4, pp. 347–370, Dec. 1991.

[23] K. Wittenburg, "Adventures in Multi-dimensional Parsing: Cycles and Disorders", in *1993 International Workshop on Parsing Technologies, Tilburg, Netherlands and Durbuy, Belgium, Aug. 8-10*, Aug. 1993.

[24] K. Marriott, "Constraint Multiset Grammars", In VL'94 [30], pp. 118–125.

[25] A.G. Cohn and J.M. Gooday, "Defining the Syntax and the Semantics of a Visual Programming Language in a Spatial Logic", in *AAAI-94, Spatial and Temporal Reasoning Workshop*, 1994, Preprint.

[26] J.D. McWhirter and G.J. Nutt, "Escalante: An Environment for the Rapid Construction of Visual Language Applications", In VL'94 [30], pp. 15–22.

[27] M. Minas and G. Viehstaedt, "DiaGen: A Generator for Diagram Editors Providing Direct Manipulation and Execution of Diagrams", In VL'95 [29], pp. 203–210.

[28] S.S. Chok and K. Marriott, "Automatic Construction of User Interfaces from Constraint Multiset Grammars", In VL'95 [29], pp. 242–249.

[29] *1995 IEEE Symposium on Visual Languages, Darmstadt, Germany, Sep. 5-9.* IEEE Computer Society Press, Sept. 1995.

[30] *1994 IEEE Symposium on Visual Languages, St. Louis, Missouri, Oct. 4-7.* IEEE Computer Society Press, Oct. 1994.

Appendix: Semantics of DL Elements

Let $\mathcal{C}$ be the set of concepts and $\mathcal{R}$ the set of roles in a DL theory. A model is a set $\mathcal{D}$ and an assignment function ξ such that $\xi : \mathcal{C} \longrightarrow 2^{\mathcal{D}}$, $\xi : \mathcal{R} \longrightarrow 2^{\mathcal{D}^2}$ where $2^{\mathcal{D}}$ is the powerset of the domain $\mathcal{D}$, where $\mathcal{D}^2 = (\mathcal{D} \times \mathcal{D})$ and where ξ must satisfy the following conditions (concept names are denoted by c and role names by r):

$$
\begin{aligned}
\xi[\text{concept name}] &\subseteq \mathcal{D} \\
\xi[\text{role name}] &\subseteq \mathcal{D} \times \mathcal{D} \\
\xi[(c_1 \wedge \ldots \wedge c_n)] &= \cap_{i=1}^{n} \xi[c_i] \\
\xi[(c_1 \vee \ldots \vee c_n)] &= \cup_{i=1}^{n} \xi[c_i]
\end{aligned}
$$

$$
\begin{aligned}
\xi[(\exists_{\geq \mathsf{n}}\ \mathsf{r})] &= \{x |\ \|\{(x,y)|\ (x,y) \in \xi[r]\}\| \geq n\} \\
\xi[(\exists_{\geq \mathsf{n}}\ \mathsf{r}\ \mathsf{c})]] &= \{x |\ \|\{(x,y)|\ (x,y) \in \xi[r] \wedge y \in \xi[c]\}\| \geq n\} \\
\xi[(\exists_{\leq \mathsf{n}}\ \mathsf{r})]] &= \{x |\ \|\{(x,y)|\ (x,y) \in \xi[r]\}\| \leq n\} \\
\xi[(\exists_{\leq \mathsf{n}}\ \mathsf{r}\ \mathsf{c})] &= \{x |\ \|\{(x,y)|\ (x,y) \in \xi[r] \wedge y \in \xi[c]\}\| \leq n\} \\
\xi[(\exists_{= \mathsf{n}}\ \mathsf{r})]] &= \{x |\ \|\{(x,y)|\ (x,y) \in \xi[r]\}\| = n\} \\
\xi[(\exists_{= \mathsf{n}}\ \mathsf{r}\ \mathsf{c})] &= \{x |\ \|\{(x,y)|\ (x,y) \in \xi[r] \wedge y \in \xi[c]\}\| = n\} \\
\xi[(\forall\ \mathsf{r}\ \mathsf{c})] &= \{x |\ \forall y : (x,y) \in \xi[r] \Rightarrow y \in \xi[c]\} \\
\xi[(=\ \mathsf{r}\ \mathsf{i})] &= \{x |\ \forall y : (x,y) \in \xi[r] \Rightarrow y = \mathsf{i}\} \\
\xi[(\mathsf{r}\bullet\ \mathsf{c})] &= \xi[r] \cap \{(x,y)|\ y \in \xi[c]\} \\
\xi[\mathsf{r}_1 \circ \mathsf{r}_2] &= \{(x,y)|\ \exists z.(x,z) \in \xi[r_1] \wedge (z,y) \in \xi[r_2]\}
\end{aligned}
$$