Spatial Graph Grammars for Graphical User Interfaces

JUN KONG
The North Dakota State University
and
KANG ZHANG and XIAOQIN ZENG
The University of Texas at Dallas

In a graphical user interface, physical layout and abstract structure are two important aspects of a graph. This article proposes a new graph grammar formalism which integrates both the spatial and structural specification mechanisms in a single framework. This formalism is equipped with a parser that performs in polynomial time with an improved parsing complexity over its nonspatial predecessor, that is, the Reserved Graph Grammar. With the extended expressive power, the formalism is suitable for many user interface applications. The article presents its application in adaptive Web design and presentation.

Categories and Subject Descriptors: D.1.7 [**Programming Techniques**]: Visual Programming; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User interfaces*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms: Human Factors, Languages

Additional Key Words and Phrases: Graph grammars, visual languages, spatial specification, visual programming, diagram parsing

1. INTRODUCTION

1.1 Background

Visual notations have been used in many disciplines and range from an architect's initial design to precise technical communication using rigorously defined

This research was partially supported by the National Science Foundation under grant number IIS-0218738.
Authors' addresses: J. Kong, Department of Computer Science and Operations Research, The North Dakota State University, ND 58105; email: jun.kong@ndsu.edu; K. Zhang, Department of computer Science, The University of Texas at Dallas, TX 75083; email: kzhang@utdallas.edu; X. Zeng, Department of Computer Science and Engineering, Hohai University, Nanjing 210098, China; email: xzeng@hhu.edu.cn.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701, USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2006 ACM 1073-0616/06/0600-0268 $5.00

notations [Chok and Marriott 2003] such as Petri nets and flowcharts, etc. As a natural offspring of string-based formal language theory, graph grammars provide a well-established foundation [Rozenberg 1997] for visual programming languages (VPLs) and a formal approach to performing computations directly on visual objects thereby inspiring wide use of visual tools and interfaces. Capable of visually specifying configurations through graph term-rewriting, graph grammars have found many applications such as software architectures [Dean and Cordy 1995] and their evolution [Métayer 1998], pattern recognition [Blosetin and Schürr 1999], and many others applications [Ehrig et al. 1999a, 1999b; Mens et al. 2002; Bottoni and Minas 2003; Heckel et al. 2003; Kreowski and Knirsch 2002]. This article focuses on a new graph grammar formalism which is application-independent and serves as a high-level visual specification language to interactive designs such as adaptive layout of multimedia documents [Qiu et al. 2003] and intelligent diagram editors [Chok and Marriott 2003].

Different from string grammars expressing sentences in sequences of characters, graph grammars are suitable for specifying visual languages expressed in a multidimensional fashion [Burnett 2006]. In other words, in addition to the left and right relationships found in a text, graphical objects in a visual program can hold various spatial relationships such as above and contain, etc. As noted by Rekers and Schürr [1996], the physical layout and the meaning of a diagram are two important aspects of a visual sentence. A *spatial relations graph* (SRG) explores spatial relationships between pictorial objects while an *abstract syntax graph* (ASG) provides structural information in a succinct form. The SRG is geared toward visualization, and the ASG toward interpretation [Rekers and Schürr 1996]. The distinction between SRGs and ASGs offers different representations of the same concept simultaneously [Bardohl et al. 1999].

1.2 Motivation

Developing expressive and intuitive graph grammar formalisms to specify VPLs has been an active research direction [Rozenberg 1997]. Few researchers, however, have introduced spatial information to the abstract syntax. Schürr [1994] proposed a triple graph grammar to specify interdependencies between graph-like structures at a high level. With one graph grammar specifying the SRGs and another defining the ASGs, the triple graph grammar maintains a loose correspondence between the abstract and spatial aspects of VPLs by introducing additional edges connecting SRG objects to the corresponding ASG objects [Rekers and Schürr 1996]. Aiming at syntax-directed layouts, Brandenburg [1995] proposed the *layout graph grammar* which directly draws rewriting rules on a plane and generates a desirable layout according to the spatial relationships defined in rewriting rules.

These approaches explore spatial relationships from the layout perspective without direct contribution to the interpretation of a graph. Picture description formalisms, such as the *constraint multiset grammar* (CMG) [Marriott 1994], are powerful in interpreting pictures through spatial relationships among visual objects. Unlike graph grammars, they do not make a strict distinction between objects (nodes) and relationships (edges). Instead, all needed spatial

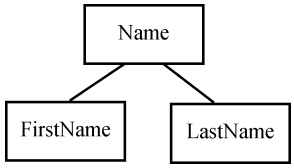


Fig. 1. A sequence in the XML schema.

or abstract relations are implicitly represented as constraints over attribute values of objects which may cause side effects that create new relationships to unknown context elements.

Due to the visual nature of VPLs, we argue that the spatial information should not only contribute to the representation, but also explicitly and intuitively convey structural and semantic information over involved objects. For example, instead of using attributes to specify an order over a collection of objects, we can visually specify the order through the spatial relationship between the involved objects (e.g., the left object has a smaller index than the right one). Figure 1 demonstrates a sequential specification in the XML schema. With traditional graph grammars, attributes or extra edges are needed to determine the sequential order of the elements *FirstName* and *LastName*. By introducing spatial information to the abstract syntax, we can visually specify the sequence of the elements through their spatial configuration without using an extra edge or attribute, for example, *FirstName* should occur before *LastName* by the default left-right order. In addition, spatial configurations can be used to specify sophisticated communications when modeling complex systems. Therefore, spatial information, when effectively used, can reduce the number of edges and provide concise representations.

This article presents a *spatial graph grammar formalism* (SGG) which introduces spatial notions into the abstract syntax and takes both the connectivity and spatial relationships among objects as the precondition of a graph transformation. The spatial extension makes the SGG suitable for specifying properties and semantic information with spatial arrangements and reduces the gap between the concrete representation of visual languages and the specification of graph grammars. Furthermore, the spatial specification can be used to narrow down the search space and derive an efficient parsing algorithm.

1.3 Overview

Computers are increasingly seen not only as computation tools but more as communication tools [Bottoni et al. 1999]. With intuitive appearances, graphs popularize visual communications where information is conveyed with a graphical representation between computers and end users. In order to perform a visual interaction properly, a computer has to understand the information carried by a graph. We focus on a visual language formalism which provides a formal basis for graphical specifications. Based on the formalism, computers can automatically perform visual communication by validating and understanding user-manipulated graphs.

Figure 2 illustrates the concept of a visual interaction framework based on the spatial graph grammar formalism. In the form of the SGG, a visual interface

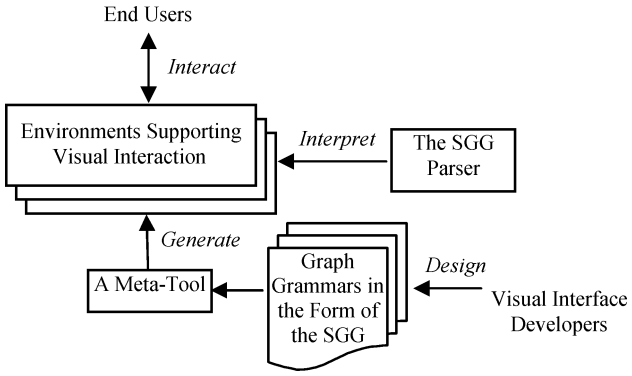


Fig. 2. Visual interaction framework based on the spatial graph grammar formalism.

developer can specify a set of graphs with valid meanings. Upon a grammatical specification, a metatool like *VisPro* [Zhang et al. 2001c] can automatically generate a graphical environment supporting visual interactions. In the generated environment, end users can directly communicate with computers by manipulating graphs. All communicated messages with a graphical appearance will be validated by the SGG parser. Therefore, based on the SGG, a visual interface developer uses a metatool to generate a graphical environment which can be used by end users.

As presented in Figure 2, the fundamental concept is the spatial graph grammar formalism together with its parser, developed from a context-sensitive graph grammar formalism, the *Reserved Graph Grammar* (RGG) [Zhang et al. 2001a]. Though the RGG is powerful in specifying structural relationships among objects, it has no support for describing what a graph looks like. The spatial graph grammar formalism (SGG) is enhanced from the RGG with a spatial extension. In general, the new formalism satisfies the following criteria.

—*Expressive*. Potential spatial relationships can be clearly specified without sacrificing the expressiveness of structural specifications.

—*Flexible*. Though tightly coupled with spatial configurations, the structural part of the grammar can be independently used, just as the original RGG, without relying on spatial specifications.

The SGG takes spatial notions as language constructs which distinguishes it from other graph grammar formalisms. Existing graph grammars model structures through nodes and edges while the SGG further introduces spatial information to the abstract syntax. In summary, the technical contributions of this article are the following.

—The SGG extends the expressive power of the RGG by introducing a spatial specification mechanism. It provides a high-level language for explicitly expressing both abstract and spatial relationships within a single grammatical definition. The expressiveness of the SGG is demonstrated by presenting an example of specifying adaptive Web interfaces.

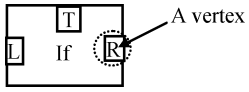


Fig. 3. A node.

—With the help of spatial specifications, the parser of the SGG can narrow down the search space and thus achieve a better performance than that of the RGG.

The rest of this article is organized as follows. Section 2 introduces the Reserved Graph Grammar formalism, the predecessor of the spatial graph grammar formalism. Section 3 presents the spatial relations that can be defined in the SGG. Section 4 formally defines the SGG formalism. Section 5 analyzes the parsing time complexity. Section 6 gives a running example, followed by Section 7 that discusses some potential applications using the proposed formalism. Section 8 reviews related work, and Section 9 concludes the article and proposes future research.

2. THE RESERVED GRAPH GRAMMAR FORMALISM

Graph grammars provide a theoretical foundation for visual languages [Rozenberg 1997]. A graph grammar consists of a set of rewriting rules which dictate the way to construct a complete graph, usually called a *host graph*, from nodes and edges. Graph grammars specify all possible legal interconnections between individual objects, that is, any edge in a valid graph can be eventually derived from a sequence of applications of rewriting rules. Conversely, an unexpected edge signals a violation on the graph grammar. A graph grammar can be used to glue various components into a complete system.

The Reserved Graph Grammar is a context-sensitive graph grammar formalism [Zhang et al. 2001a] which is expressive in specifying various types of graphs. The RGG formalism is expressed in a node-edge format, similar to the box-and-line drawings [Allen and Garlan 1994] (or node-link diagrams [Irani and Ware 2003]) to suit automatic analyses through graph grammars. In an RGG, nodes are organized into a two-level hierarchy, where a large rectangle representing the node itself is the first level, and embedded small rectangles called vertices are the second level. Figure 3 depicts a typical RGG node which includes various vertices. In a node, each vertex is uniquely identified. A node can be viewed as a module, a procedure, or a variable, etc., depending on the design requirement and granularity. A vertex functions as the connecting point attached to an edge. Edges are used to denote communications or relationships between nodes.

Based on nodes and edges, the RGG offers a formal approach to specifying the evolution of graphs. In general, a graph grammar includes a set of rewriting rules, and each rule consists of two subgraphs, called the *left graph* and the *right graph*. The application of a rule to a host graph, that is, a graph transformation, replaces a subgraph in the host graph that matches the left (or right) graph of the rule by the right (or left) graph. The RGG uses the *marking* technique which classifies vertices as marked and unmarked ones to address the embedding

issue, that is, building connections between the replacing of the subgraph and the surrounding of the replaced subgraph in the host graph. A marked vertex is identified by a unique integer and preserves its associated edges connected to nodes outside a replaced subgraph.

The RGG provides a means of associating data to a node in terms of *attributes*. An attribute expresses a piece of data related to the object represented by a node and can be retrieved and evaluated in the process of parsing. Therefore, a graph represented in RGG notations can be executable.

The RGG is equipped with a deterministic parsing algorithm, called the *selection-free parsing algorithm* (SFPA) [Zhang 1998]. A graph grammar must satisfy the selection-free condition in order to use SFPA. Informally, the selection-free property ensures that different orders of applications of rewriting rules produce the same result. We developed an algorithm to automatically check whether a graph grammar satisfies the selection-free condition [Zhang et al. 2001a]. Though it is unclear how this condition limits the application scope, it is interesting to note that even grammars for some complicated graphs satisfy the condition [Zhang et al. 2001b]. We proved that a failed parsing path indicates an invalid graph, and thus SFPA is efficient with a polynomial parsing complexity by only trying one parsing path [Zhang et al. 2001a].

3. SPATIAL RELATIONSHIPS

A graph grammar precisely defines a class of valid configurations at a certain abstract level. Nodes and edges represent objects and relationships, respectively. Graph grammars have the inherent ability to express visual language in a multidimensional fashion, and the SGG further incorporates spatial notions into language constructs. This section introduces spatial relationships that can be qualitatively defined in the SGG.

A *qualitative representation*[1] of spatial information utilizes discrete quantity space in which distinctions are relevant to the application domain. In other words, distinctions are only introduced if they are necessary to model some particular aspects of the domain [Cohn 1997]. Many spatial models have been proposed to qualitatively represent spatial information from different aspects. Based on some well-established models, the SGG allows visual interface developers to specify qualitative spatial information in different application domains.

In general, the description of a scene of objects in space involves spatial aspects that have an expression both in terms of inherent characteristics of each object and in the context of other objects [Clementini et al. 1997]. The size and shape of an object illustrate its own properties while spatial relations express configurations among distinct spatial objects. Spatial relations, which the SGG takes as language constructs, can be classified into several types [Pullar and Egenhofer 1988] including *direction* relations that describe an order in space, *topological* relations that express neighborhood and incidence, and *distance* relations such as near and far. Furthermore, *alignment* relations are introduced to illustrate projections of two objects on the x/y-axis.

[1]The term qualitative is widely used in the spatial cognition community, for example, Clementini et al. [1997]; Cohn [1997]; Cohn and Hazarika [2001]; and Frank [1996].

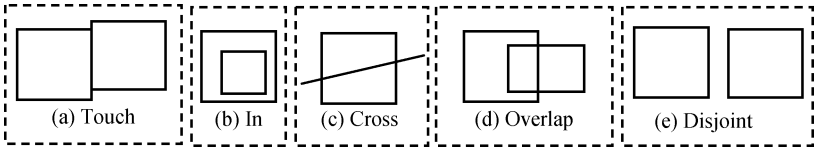


Fig. 4. Topological illustrations.

3.1 Topology

Topology is perhaps the most fundamental aspect of space [Cohn and Hazarika 2001], and topological relations are able to describe all aspects of the scene which are invariant with respect to common linear transformations (e.g., translation, rotation) [Clementini et al. 1997]. In the SGG, topological relationships are defined over objects satisfying the following properties:

—an area connected with no holes; and

—a line that has no self intersections, and is either circular or has only two endpoints.

By viewing an object λ as a point set, topological relationships are classified according to how two objects intersect with each other. In order to express various intersecting cases, an object is subdivided into the *boundary*[2] and *interior*, denoted as $\partial\lambda$ and λ°. Using the *dim*[3] function which determines the dimension of a point set, Clementini et al. [1993] grouped related cases together and summarized five fundamental and mutually exclusive topological relationships, that is, *touch*, *in*, *cross*, *overlap*, and *disjoint*. Informally, two objects touch each other if they share common points only on the boundary. One object is in another if the points of the former are completely contained in the latter. The cross relationship explores the intersecting case between two lines or between a line and an area. A line crosses another if they have a common internal point; a line crosses an area if the line penetrates the interior of the area. The overlap relationship illustrates the intersection between two homogeneous objects, that is, between a pair of lines or areas. Different from touch and cross, overlap requires the intersecting part to be of the same dimension as the involved objects. For example, two lines overlap each other only if their intersection is a line. Figure 4 illustrates a visual example for each relationship. The five relationships are formally defined as the following [Clementini et al. 1993]:

—λ_1 touches with λ_2 iff $(\lambda_1^\circ \cap \lambda_2^\circ = \emptyset) \wedge (\lambda_1 \cap \lambda_2 \neq \emptyset)$;

—λ_1 is in λ_2 iff $(\lambda_1^\circ \cap \lambda_2^\circ \neq \emptyset) \wedge (\lambda_1 \cap \lambda_2 = \lambda_1)$;

—λ_1 is cross with λ_2 iff $\dim(\lambda_1^\circ \cap \lambda_2^\circ) = (\max(\dim(\lambda_1^\circ), \dim(\lambda_2^\circ)) - 1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2)$;

[2]The boundary of a point is always empty; the boundary of a line, is empty in the case of a circular line, while otherwise it is the set of the two separate endpoints; the boundary of an area is the circular line consisting of all the accumulation points of the area [Clementini et al. 1993].

[3]S is a general point set, and the dim function is defined as the following [Clementini et al. 1993]:

$$\dim(S) = \begin{cases} -1 \text{ if } S = \emptyset; \\ 0 \text{ if S contains at least a point, and no lines or areas;} \\ 1 \text{ if S contains at least a line and no areas;} \\ 2 \text{ if S contains at least an area.} \end{cases}$$

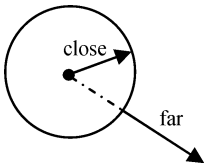


Fig. 5. The first level of granularity.

—λ_1 overlaps with λ_2 iff $(\dim(\lambda_1^\circ) = \dim(\lambda_2^\circ) = \dim(\lambda_1^\circ \cap \lambda_2^\circ)) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2)$;
—λ_1 is disjoint with λ_2 iff $\lambda_1 \cap \lambda_2 = \emptyset$.

Developers can easily elaborate the five topological relationships to finer granularities as required by the application. For example, a total of 52 topological relationships defined in the *dimension extended approach* can be represented by the logical conjunction/disjunction of those five basic relationships [Clementini et al. 1993].

3.2 Direction

In addition to topological relationships, direction relations are another important aspect describing spaces where we approximate objects as points. A direction describes the location of a primary object relative to the reference object and is viewed as a binary function that maps two points onto one element belonging to a set of symbolic directions, for example, D_4 = {N, E, S, W} or more extensively, D_8 = {N, NE, E, SE, S, SW, W, NW}. The SGG adopts the cone-based approach [Peuquet and Zhan 1987] that partitions a plane into cone-shaped areas and relates the angular direction between the primary object and the reference object to the nearest named direction. For example, *north* may refer to a cone-shaped region of 45° with an axis pointing to the true north in D_8. With a composition table [Frank 1996], one can deduce a new direction from a pair of known directions.

3.3 Distance

A set of distance relations partitions the space into regions centered on the reference object. A distance distinction is made by comparing magnitudes of distances and assigning to a distance an appropriate distance relation. In general, at a given granularity, space surrounding a reference object is partitioned according to a number of totally ordered distance distinctions $Q = \{q_0, q_1, \ldots, q_n\}$, where q_0 is the distance closest to the reference object, and q_n is the one farthest away (to infinity) [Hernández et al. 1995]. For example, the first level of granularity distinguishes *close* from *far* as presented in Figure 5, that is, Q = {*close*, *far*}. The two distance relationships divide the plane into two regions where the outer region goes to infinity. Each distance distinction q_i is associated with a distance range δ_i. If the distance between two objects falls in δ_i, they have a distance relationship q_i.

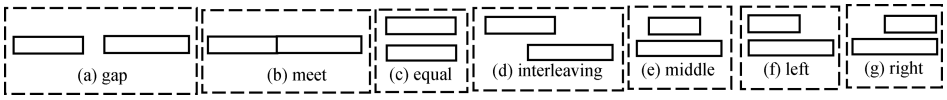


Fig. 6. Alignment relationships.

3.4 Alignment

The comparison of projections of objects on the x/y-axis provides additional spatial information, and thus *alignment* relationships are introduced to supplement direction, topology, and distance. Allen [1983] proposed intervals of time with thirteen relations, which are also applicable in the spatial context. Ignoring six inverse relations the SGG distinguishes seven types of alignment relationships, that is, *gap*, *meet*, *equal, interleaving, middle, left* (*bottom* if in the vertical direction), and *right* (*top*), which are specified either in the vertical direction or in the horizontal direction. The following description only illustrates alignment relationships in the horizontal direction, and the same principle applies to those in the vertical direction with x coordinates being replaced by y coordinates. *Gap* defines a spatial interval as coinciding with the x-axis direction that exists between the greatest lower boundary of one object and the least upper boundary of another; *meet* defines the greatest lower boundary of one object that just meets the least upper boundary of another; *equal* indicates two objects having the same projection on the x-axis; *interleaving* means that the projections of two objects on the x-axis are overlapping; *middle* denotes the projection of one object within the range of another object's projection; *left* or *right* specifies that two objects share the same greatest lower boundary or the least upper boundary. Figure 6 depicts these relationships.

3.5 Spatial Granularity and Spatial Hierarchy

Different application domains may require different granularities of spatial relationships. For example, considering the direction aspect, only two distinctions, that is, left and right, are necessary to distinguish a sequential order. On the other hand, more elaborate distinctions are needed to specify the relative positions in the application of multimedia layout. Based on the previous spatial models, developers can define spatial relationships at a granularity suitable for their domain applications from four aspects, that is, the direction, topology, distance and alignment. Spatial relationships may not be necessarily orthogonal. For example, an equal relationship (in vertical direction) in the alignment aspect satisfies the spatial property of the top relationship. In other words, a top relationship can be derived from an equal relationship. Developers can, therefore, organize spatial relationships hierarchically. A super type (e.g., the top relationship) represents general information, while a subtype (e.g., the equal relationship) indicates specific information and preserves the spatial property of the super type. Denoted as $r_2 \leq r_1$, a relationship r_1 *preserves* the spatial property of another relationship r_2 if $\forall x, y\, R_1(x,y) \rightarrow R_2(x,y)$[4], where R_1 and R_2 associated with relationships r_1 and r_2 are two boolean expressions

[4]In this context, the notation "$\rightarrow$" denotes the logic connective of implication.

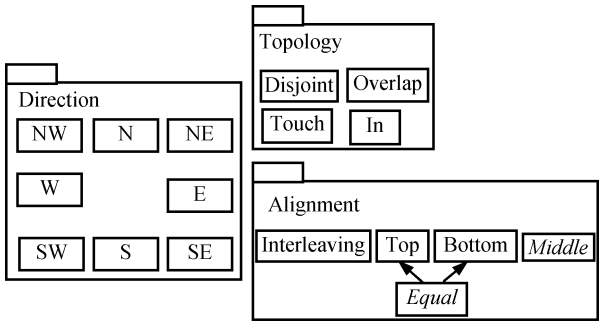


Fig. 7. Spatial relationships.

for verifying the spatial property between objects x and y. For example, we can conclude that the equal relationship preserves the top relationship, that is, top $\leq$ equal. A textual definition in the form of $< x, r, y >$ represents that the spatial relationship r is held over two objects, x and y. Associating a list of such definitions with an RGG rewriting rule, an SGG rule expresses both spatial and abstract structures.

Developers may design a set of graphical notations to represent spatial relationships, and every SGG rule can have an equivalent graphical representation when replacing textual definitions with corresponding graphical notations. We have previously proposed a set of spatial relationships (as shown in Figure 7 where directed edges denote the hierarchy of spatial relationships) and a corresponding set of graphical notations (as illustrated in Figure 8) for adaptive Web interfaces [Qiu et al. 2003]. In this set of graphical notations, a node is divided into nine areas, each except the central area indicating the relative direction of the node connected to it. Moreover, shapes of boundaries represent various topological and alignment relationships.

3.6 Syntax-Directed Computation

In the SGG, the application of a rewriting rule may create new objects or adjust the current spatial configuration. The SGG applies *action codes* to derive new spatial properties from current spatial attributes. The syntax-directed computation supplements qualitative specifications with quantitative information by manipulating node attributes which record the spatial properties of an object. Like in the RGG, action codes are associated with rewriting rules and are triggered when a rewriting rule is applied. Writing an action code is like writing a standard event handler in Java. Action codes are suitable for specifying spatial reconfigurations in the course of graph transformation. For example, to merge two objects into a larger one as shown in Figure 9(a), we can associate the following action code to the rewriting rule in Figure 9(b).

```
Action(AAMGraph g)
{
        R.top = P.top;
        R.Left = P.left;
```

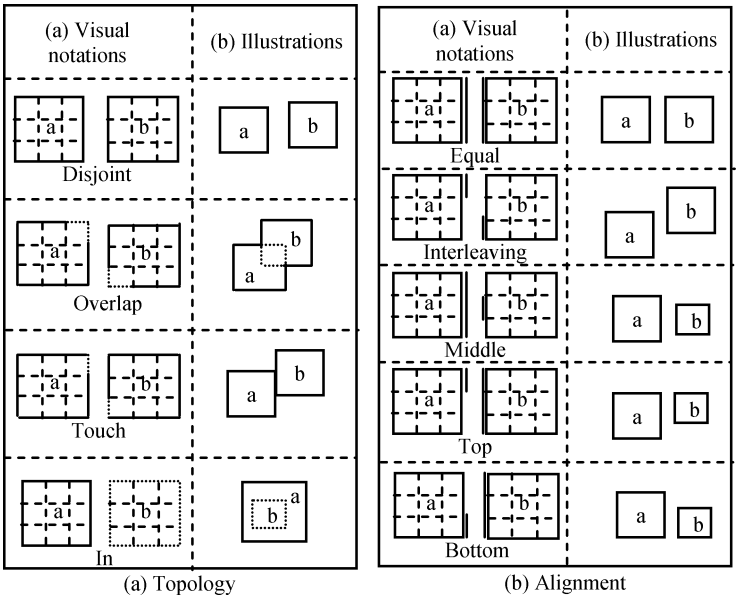


Fig. 8. Graphical notations.

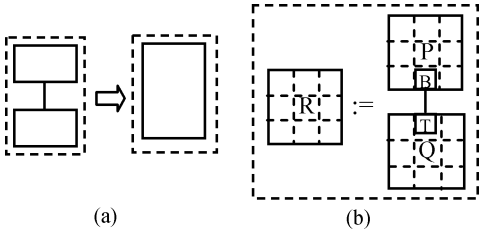


Fig. 9. Using action codes to support spatial reconfiguration.

```
        R.height = P.top − Q.bottom;
        R.width = P.width;
}
```

4. SPATIAL GRAPH GRAMMARS AND THEIR LANGUAGES

This section formally defines the spatial graph grammar formalism (SGG), extended over the Reserved Graph Grammar [Zhang et al. 2001a].

4.1 Preliminaries

In a spatial graph grammar, nodes are used to represent objects. Unlike other graph grammars, the SGG views a node as a collection of vertices, which denote the ports connecting to other objects. Therefore, the term *node* has a different meaning from the term *vertex*.

Definition 1. Given a vertex label set Ω^N, a *node* is a tuple $N = \langle V^N, l^N, name^N \rangle$, where V^N is a set of vertices, $l^N : V^N \to \Omega^N$ is an injective[5] function labeling vertices, and $name^N$ denotes the node label.

The injective function l^N in Definition 1 specifies that a vertex in a node is uniquely identified within a node by its vertex label. For example, the *If* node in Figure 3 contains three vertices named by the vertex label set $\Omega^N = \{T, L, R\}$. A node label denotes the class of node instances. Nodes with the same label are made of the same set of vertices.

Denoted as $v_1 \approx v_2$, two vertices v_1 and v_2 are *isomorphic* if they have the same vertex label within nodes of the same node label. Since the SGG organizes nodes in a two-level hierarchy, comparing two nodes involves not only matching the node labels, but also verifying if their vertices are isomorphic.

Definition 2. Over two nodes N and N', the *structural isomorphism* g: $V^N \to V^{N'}$ is a bijective[6] function which preserves the labels of vertices, that is, satisfying $l^{N'} o\, g \approx l^N$.[7] Two nodes are *isomorphic*, denoted as $N \approx N'$, iff they have structural isomorphism and the same node label.

Definition 3. Over a set of objects O, a *spatial signature* is a function $f : O \times O \to Top \times Dir \times Dis \times Align$, where the sets *Top*, *Dir*, *Dis* and *Align* represent topological, direction, distance, and alignment relationships, respectively. In particular, every set has a special element denoted as *none*[8], indicating that no spatial relationship specified between two objects. f maps a pair of nodes to a spatial configuration expressed through spatial relationships of the four spatial aspects.

A spatial signature formally identifies a qualitative spatial representation which makes up the precondition of a graph transformation with the connection among nodes. For example, with the right graph of the rewriting rule in Figure 10(a), we can conclude a spatial signature in Figure 10(b) according to Definition 3.

Definition 4. A *graph* is a tuple $G := \langle N^G, E^G, s^G, t^G, g^G \rangle$, where N^G is the set of nodes, E^G is the set of edges, $N^G.V^G$ is the set of vertices constructing N^G, s^G: $E^G \to N^G$. V^G and t^G:$E^G \to N^G.V^G$ are two functions that specify the source and target points of an edge, g^G is the spatial signature defined on the node set N^G.

According to Definition 4, the SGG uses nodes (N^G), edges (E^G), and a spatial configuration (g^G) to specify the language. More specifically, nodes represent objects, and edges glue different objects into a complete graph. Furthermore, the spatial configuration among objects also explicitly contributes to the interpretation of graphs.

---

[5]A function $f: X \to Y$ is injective if $f(x) = f(y)$ implies $x = y$.

[6]A bijective function is a total function that is both injective (one-to-one) and surjective (onto). More formally, a function $f: X \to Y$ is bijective if, for every y in the codomain Y, there is exactly one x in the domain X with $f(x) = y$.

[7]Functions can be combined using the composition operation denoted by o. For $f : R \to S$ and $g : S \to T$, $g\,of$ is the function with domain R and codomain T such that for all $x \in R$, $g\,of(x) = g(f(x))$.

[8]The *none* is the least value in any ordering over a value set.

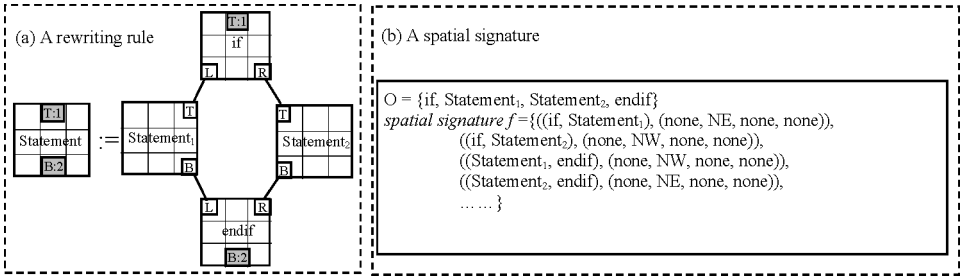


Fig. 10. A spatial signature.

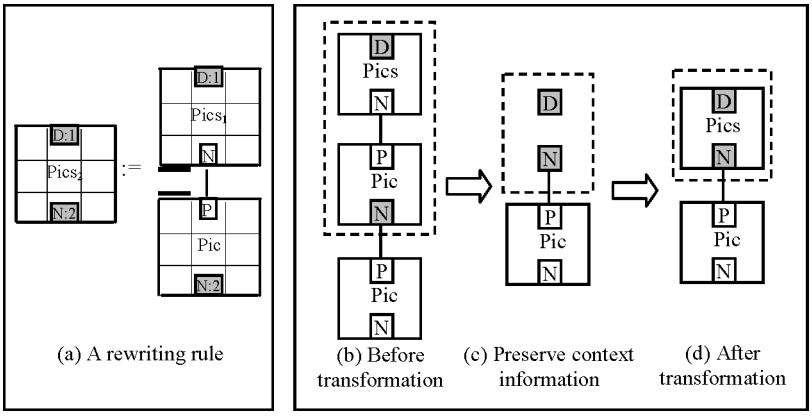


Fig. 11. Marked vertices in preserving the context information.

Due to the multidimensional nature of visual languages, the SGG uses the *marking* technique [Zhang et al. 2001a] to address the embedding issue, that is, building connections between a replacing subgraph and the surrounding of a replaced subgraph. In a rewriting rule, a vertex is *marked* by postfixing its label with a unique integer. For example, in the rewriting rule of Figure 11(a), vertex D of $Pics_1$ is marked, while the vertex N is unmarked. If a vertex v in a replaced subgraph maps to a marked vertex, v will be preserved during graph transformation to establish connections between the surrounding of a replaced subgraph and a new subgraph. For example, Figure 11(b) shows an isomorphic subgraph (in the dotted rectangle) corresponding to the right graph of the rewriting rule in Figure 11(a), and highlights with gray background the vertices mapping to marked vertices. Those highlighted vertices will be preserved in a graph transformation as shown in Figure 11(c), and the transformed graph is shown in Figure 11(d).

Definition 5. mark: V$\rightarrow$I is a partial and injective function[9], where V is a set of vertices and I is a set of integers. A vertex v is *marked* if and only if mark(v) $\downarrow$.

[9]A partial function on a set V is simply a function whose domain is a subset of V. If f is a partial function on V and $v \in V$, then we write $f(v) \downarrow$ and say that $f(v)$ is defined to indicate that v is in the domain of f. If v is not in the domain of f, we write $f(v) \uparrow$ and say that $f(v)$ is undefined.

In the SGG, a marked vertex v_1 in the right graph of a rewriting rule has a counterpart v_2 in the left graph. In other words, if v_1 in the right graph and v_2 in the left graph are marked with the same integer, they map to the same context element in a graph transformation. We use the notation $v_1 \equiv v_2$ to represent that the marked vertex v_1 in the right graph has the same marking integer as the marked vertex v_2 in the left graph.

Definition 6. A *marked graph* is a tuple $G := \langle N^G, E^G, s^G, t^G, g^G, mark^G \rangle$. The first five elements are the same as those in Definition 4, and $mark^G$ is a marking function.

4.2 Graph and Spatial Morphisms

The basic concepts have been introduced and, this section defines the notion of comparisons between graphs.

Definition 7. A *graph morphism* $f: G \rightarrow G'$ defines a pair of bijective functions $\langle f^N: N^G \rightarrow N^{G'}, f^E: E^G \rightarrow E^{G'} \rangle$ that satisfy:

(1) $s^G \approx s^{G'} o\ f^E$;
(2) $f^N o\ abs^G o\ s^G = abs^{G'} o\ s^{G'}\ of^E$;[10]
(3) $t^G \approx t^{G'} o\ f^E$;
(4) $f^N o\ abs^G o\ t^G = abs^{G'} o\ t^{G'} o\ f^E$.

A graph morphism specifies a mapping of nodes and edges between two graphs. In Definition 7, the first two requirements state that if an edge e in the graph G maps to an edge e' in G', the vertex v inside a node n attached to e in G should be isomorphic to a vertex v' inside a node n' attached to e' in G'. The other two requirements illustrate the other connecting point.

Introducing spatial information to the abstract syntax, the comparison of two graphs involves not only connections between nodes (defined as a graph morphism in Definition 7), but also spatial configurations (a spatial morphism in Definition 8). Denoted as $s_2 \leq s_1$, a spatial configuration s_1 between a pair of objects *preserves* the spatial property of another configuration s_2 if every spatial relationship in s_2 is preserved in s_1. In order to extract the relationship in each of the four spatial aspects from a spatial signature of a graph G, we define the following four high order functions:

$$P_1: O \times O \times g^G \rightarrow Top;\ P_2: O \times O \times g^G \rightarrow Dir;$$
$$P_3: O \times O \times g^G \rightarrow Dis;\ P_4: O \times O \times g^G \rightarrow Align.$$

For example, with the spatial signature in Figure 10, P_2 extracts the direction relationship between two objects, for example, $P_2(\text{if}, \text{Statement}_1, g^G) = \text{NE}$.

Definition 8. Over graphs G and G', a *spatial morphism* $sp: N^G \rightarrow N^{G'}$ is a function that maps nodes between G and G' so that the spatial configuration between any pair of objects in G' preserves the spatial properties between the

[10]With a graph G, the function $abs^G: N^G.V^G \rightarrow N^G$ maps each vertex to a node such that the vertex belongs to the node.

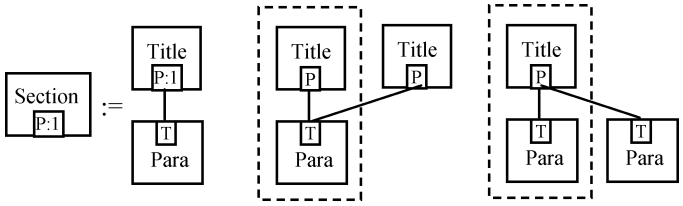


Fig. 12. The usage of marking technique in graph transformation.

corresponding pair of nodes in G, that is, satisfying $\forall n_1, n_2 \in N^G$: $g^G(n_1,n_2) \leq g^{G'}(sp(n_1), sp(n_2))$.

Based on graph morphism and spatial morphism, Definition 9 specifies a *redex* which denotes in a given graph (i.e., a host graph) a subgraph matching another given graph.

Definition 9. A subgraph X of a graph H is called a redex of a marked graph G, denoted as $X \in \text{Redex}(H, G)$, if and only if

(1) $f = \langle f^N: N^G \rightarrow N^X, f^E: E^G \rightarrow E^X \rangle$ is a graph morphism between G and X;
(2) f^N also serves as a spatial morphism between G and X;
(3) $\forall v \in N^G.V^G, \forall e' \in E^H$: $(((s^H(e') \approx v) \wedge abs^H(s^H(e')) = f^N(abs^G(v))) \vee ((t^H(e') \approx v) \wedge abs^H(t^H(e')) = f^N(abs^G(v)))) \wedge mark^G(v) \uparrow \Rightarrow e' \in E^X$

The first two conditions state that X and G are isomorphic in structure and have a spatial morphism. The third condition states that if a vertex in a rewriting rule is unmarked and maps to a vertex v in the redex of a host graph, then all edges connected to v should be completely inside the redex. Figure 12 illustrates the third condition. Assume that a *Title* node may connect to multiple *Para* nodes, while a *Para* node is allowed to connect to only one *Title* node since a title may be followed by several paragraphs but not vice versa. Such a restriction is easily expressed by marking vertex P of node *Title* and leaving T of node *Para* unmarked in the definition in Figure 12(a). According to the third condition, the isomorphic graph (enclosed in a dotted rectangle) in Figure 12(b) is not a redex because vertex T in *Para* has an edge outside the subgraph while its isomorphic vertex in the right graph is unmarked. On the other hand, Figure 12(c) includes a legal redex (enclosed in a dotted rectangle) because the vertex P of *Title* has its isomorphic vertex marked in the right graph and thus allows edges connected outside the subgraph.

4.3 Graph Transformation and Graph Languages

We now proceed to define the graph transformation process. A graph grammar is made up of a set of rewriting rules called productions. Each production consists of two graphs, called left graph (L) and right graph (R). When a production is applied to a host graph, the host graph is said to be transformed by the application.

Definition 10. A *production* p: = (L, R) is a pair of marked graphs, that is, left graph L and right graph R over the same node label set.

Definition 11. Let X in H be the redex of G, that is, $X \in \text{Redex}(H, G)$. The *transformation* from H to H' by replacing X with G', denoted as $H' = \text{Tr}$ *(H, G, G', X)*, is defined as:

(1) Add G' to H;
(2) Find all edges in H such that each edge's one side, denoted as c_1, connects to a vertex in X isomorphic to a marked vertex in G, and the other side, denoted as c_2, connects to a vertex outside X;
(3) Redirect c_1 to the marked isomorphic vertex in G' and keep c_2 unchanged;
(4) Delete X from H; and
(5) Set the coordinates and sizes for nodes in G' according to the computation defined in the action code.

A graph transformation in the SGG not only updates the graph configuration but also computes attributes for spatial evolution. The spatial signature, spatial morphism, and action code cooperate to complete a process of spatial evolution:

—a spatial signature summarizes the spatial property of a graph;
—working on spatial signatures, a spatial morphism, together with a graph morphism, makes up the production application condition; and
—an action code calculates the spatial reconfiguration and results in a new spatial signature.

Based on the definition of graph transformation, the *L-application* and *R-application* can be correspondingly defined as follows.

Definition 12. An *L-application* of a production p := (L, R) to a graph H is a transformation $H' = \text{Tr}(H, L, R, X)$, where $\text{X} \in \text{Redex}(H, L)$, denoted as $\text{H} \mapsto^{\text{X}} \text{H'}$.

Definition 13. An *R-application* of a production p = (L, R) to a graph H is a transformation $H' = \text{Tr}\ (H, R, L, X)$, where $\text{X} \in \text{Redex}(H, R)$, denoted as $\text{H} \rightarrow^{\text{X}} \text{H'}$.

Definition 14. A *spatial graph grammar gg* is a tuple (*A, P, T, N*), where A is an initial graph, P is a set of graph grammar productions, T is a set of terminal labels, N is a set of nonterminal labels. For $\forall p = (L, R) \in P$ and $\forall\, l \in \text{T} \cup \text{N}$:

(1) $R \neq \text{NULL}$;
(2) L and R are over the the same label set $\text{T} \cup \text{N}$;
(3) $l \in L_i$ where $L_i \subset \{\text{L}_0, \ldots, L_n\}$ is a global layer set and $\text{L}_0 \cap \cdots \cap \text{L}_n = \text{NULL}$; and
(4) $L < R$ with respect to the following order of graphs: $G < G'$ iff $\exists$i: $|G|_i < |G'|_i \wedge \forall j < i : |G|_j = |G'|_j$ with $|G|_k$ defined as $|\{\text{x} \in G|\ \text{layer(x)} = \text{k}\}|$.

The layering condition imposes an order over node labels, which guarantees the termination of the parsing process [Rekers and Schürr 1997; Zhang et al.

2001a]. We follow the approach of Rekers and Schürr [1997] for automatically computing the layers of labels: layer(x) $\geq$ layer(y) for any x $\in$ p.L and y $\in$ p.R, where p.L and p.R represent distinct nodes in the left and right graph of a production p. This rule determines the assignment of labels to layers completely under the additional assumption that layer(x) $\geq$ layer(y) => layer(x) > layer(y) whenever possible.

We denote the sequence of intermediate derivations $H \mapsto^{X1} H_1, H_1 \mapsto^{X2} H_2, \ldots, H_{n-1} \mapsto^{Xn} H_n$ as $H \mapsto^{X1 \cdots Xn} H_n$, or simply $H \mapsto^* H_n$, where n may be zero in which case $H = H_n$.

Definition 15. let $gg = (A, P, T, N)$ be a spatial graph grammar. Its language L is defined by $L(gg) = \{G \mid A \mapsto^* G$, where G contains only elements with terminal labels$\}$.

Starting from an initial graph, iterative applications of productions generate all graphs belonging to a graph grammar. In other words, a graph grammar defines a visual programming language which consists of a set of visual sentences manipulated by end users.

This section has defined the spatial graph grammar formalism. Different from traditional graph grammars that treat spatial relations as uninterpreted abstract relations [Marriott and Meyer 1997], the SGG introduces spatial notions to the abstract syntax. Extended from the RGG, the SGG allows developers to integrate structural and spatial relations in a uniform framework. Without using the extended spatial specifications, the SGG has the same expressive power as the RGG which has found many applications [Zhang et al. 2001b, 2001c]. A full set of possible graphs defined by an SGG grammar constructs a language of the grammar. Even for string grammars, it is undecidable whether an arbitrary context-sensitive grammar produces an empty language or an infinite language [Révész 1983]. Therefore, it is undecidable whether an arbitrary SGG grammar produces an empty or an infinite language.

5. GRAPH PARSING

The context-sensitive spatial graph grammar formalism has been defined, and this section presents a parsing algorithm which uses spatial specifications to narrow down the search space and then analyzes the time complexity.

The parsing process is a sequence of R-applications which is modeled as recognize-select-execute [Bardohl et al. 1999]. One or more occurrences of a right graph may exist in the host graph, and the selection will affect the parsing result. Even for the most restricted classes of graph grammars, the membership problem is NP-hard [Rozenberg and Welzl 1986]. To allow efficient parsing without backtracking, we are only interested in *confluent* graph grammars. Informally, the confluence requires that different orders of applications of productions achieve the same result. Though it is unclear how this condition limits the application scope, it is interesting to note that even grammars for some complicated graphs satisfy the condition [Zhang et al. 2001b]. According to Fischer et al. [1998], many real world applications do not include specifications requiring backtracking, evidenced by over 1000 pages of specifications using PROGRES [Schürr et al. 1999]. Figure 13 illustrates a parsing algorithm

```
Parsing (HostGraph host)
{
 while (host!= An initial graph A)
 {
  matched = false;
  for all p∈ Productions
  {
   Redex = FindRedexForR(host, p);
   If (Redex != NULL)
   {
    R-application(host, p, Redex);
    Matched = true;
   }
  }
  if (matched == false)
  {
   print ("Invalid");
   exit (0);
  }
 }
}
```

Fig. 13. The parsing algorithm.

for confluent grammars which only tries one parsing path. In other words, if one parsing path fails, other parsing paths will also fail.

Taking advantage of spatial specifications in productions, the SGG parsing algorithm sequences objects according to their spatial configurations and thus narrows down the search space to achieve a better performance than that of its predecessor, that is, the RGG. The following sections explore the parsing algorithm in detail.

5.1 Matching Algorithm

Searching for a redex in a host graph becomes the key to the parsing process when the parser need not care about the application order. Without an order imposed on objects in the original Reserved Graph Grammar (RGG), searching for m objects in a host graph G runs in $O(m^2|G|^m)$ time [Zhang et al. 2001a]. With the spatial information found in the host graph and productions, we should be able to narrow down the search space by sequencing objects. Based on spatial specifications between objects in the right graph of a production, we generate a sequence, called a *pattern sequence*. With the same criteria to generate the pattern sequence, objects in a host graph are sequenced to an ordered list, called a *host sequence*. Efficient string-matching techniques are applied to searching for the pattern sequence in the host sequence, and then a match of the pattern sequence is extended to a redex.

Figure 14 shows the algorithm *FindRedexForR* to find a valid redex in a host graph according to a given production. It proceeds as follows:

(1) encode the objects in the right graph of the production into a sequence, that is, the *pattern sequence*;
(2) encode the objects of the host graph into a sequence, that is, the *host sequence*;

```
Redex FindRedexForR(HostGraph G, Production P)
{
   R=SequenceRightGraph(P.R);
   H=SequenceHostGraph(G, P);
   Index=SequenceIndex(R);
   redex = match(H, R, Index, P, G);
   return redex;
}
```

Fig. 14. The *FindRedexForR* algorithm.

```
Redex match (Sequence H, Sequence R, Link Index, Production P, HostGraph G)
{  i=1;  redex=NULL;
  while (i≤|G|)
  {  for all k∈ Index[H[i]]
        {  UpdateSet(k);
           if (k==|R|)
             redex = VerifyNPObj(P, G);
           if (redex != NULL)
             return redex; }
     i++
  }
  return redex;
}
```

Fig. 15. The *Match* algorithm.

(3) search for the pattern sequence in the host sequence as illustrated in the *match* algorithm in Figure 15.

We first generate a directed acyclic graph (DAG) based on the spatial specifications of the objects in the right graph of a production, and then search in the DAG for the path containing the maximal number of nodes. In a DAG, a node represents an object in the right graph and a directed edge denotes a north-south or west-east relationship between objects specified in a production. A DAG denoting north-south/west-east relationship is called V-DAG/H-DAG. As the first step of *FindRedexForR* (Figure 14), *SequenceRightGraph* first generates a V-DAG and an H-DAG, and then searches for the path with the maximal number of nodes. The objects in the longest path construct the pattern sequence. *SequenceRightGraph* proceeds as follows (the description only illustrates operations on a V-DAG, and the same principle applies to an H-DAG).

(1) Create nodes for objects in the right graph: each unique object in the right graph of a production is represented by a node.
(2) Interconnect nodes. In the right graph, if an object is defined locating southwest, south, or south-east to another object, a directed edge is added to connect the two corresponding nodes from north to south.
(3) Search for the longest path. Search for the path containing the maximal number of nodes.

Comparing the two longest paths in the V-DAG and the H-DAG, select the longer one to construct the pattern sequence. An object in the pattern sequence is called a *pattern object*; otherwise, it is a *nonpattern object*. The objects in the pattern sequence are arranged in descending order of their

y-coordinates if the longer path comes from the V-DAG, or in ascending order of their x-coordinates if the longer path comes from the H-DAG. This article assumes that the origin locates at the left-bottom corner of the screen, and the y-axis extends to the north, while the x-axis extends to the east.

The same criterion is applied to sequence the host graph. More specifically, if the pattern sequence comes from the V-DAG, object a with the largest y-coordinate is ordered before object b with a smaller y-coordinate in the host graph (a's index $<$ b's index). If two objects have the same y-coordinates, their order in the host sequence is determined by their x-coordinates, and the object with a smaller x-coordinate holds a smaller index. On the other hand, objects in the host graph are sequenced according to x-coordinates if the pattern sequence is derived from the H-DAG. The coordinate of the central point is used to represent the physical position of the object in the sequencing process.

By sequencing host graphs and productions, the problem of searching for a redex in a host graph becomes that of searching for a pattern sequence in a host sequence. Once a match is found, we can extend the match to a redex by incorporating nonpattern objects as processed in the function *VerifyNPObject* in Figure 15.

In the *Match* algorithm in Figure 15, corresponding to a pattern sequence containing m objects, $(m-1)$ sets are used to record subsequences of the host sequence. In particular, each subsequence in the kth ($k < m$) set satisfies the following requirement: the subsequence includes k objects, which have a morphism to the first k objects in the pattern sequence. Initially, all sets are empty. Once a new subsequence is found, it is inserted into a corresponding set. Furthermore, each object of the host sequence maintains a set of pointers, pointing to the subsequences that include the object. From the first object of the host sequence, the *Match* algorithm proceeds as follows:

(1) Assume that the class of the current object a (we treat the type of a node as its class) takes the ith position in the host sequence and the kth position in the pattern sequence. Extend every subsequence s in the $(k-1)$th set with object a. If an extended subsequence has a morphism to the first k objects in the pattern sequence, insert the extended subsequence into the kth set. When searching for a morphism between an extended subsequence and the pattern sequence, we only need to check those relationships associated with object a since other relationships have been verified when s is inserted into the $(k-1)$th set.

(2) If the class of object a can take other positions in a pattern sequence, repeat Step (1) to investigate other positions.

(3) Move to the next object in the host sequence, and go to Step (1).

(4) Whenever a pattern sequence is found in the host sequence, search the nonpattern objects in the remaining objects of the host sequence. If the nonpattern objects and pattern objects have a morphism to the right graph of a production, a redex made up of nonpattern objects and pattern ones is found, and a graph transformation is performed.

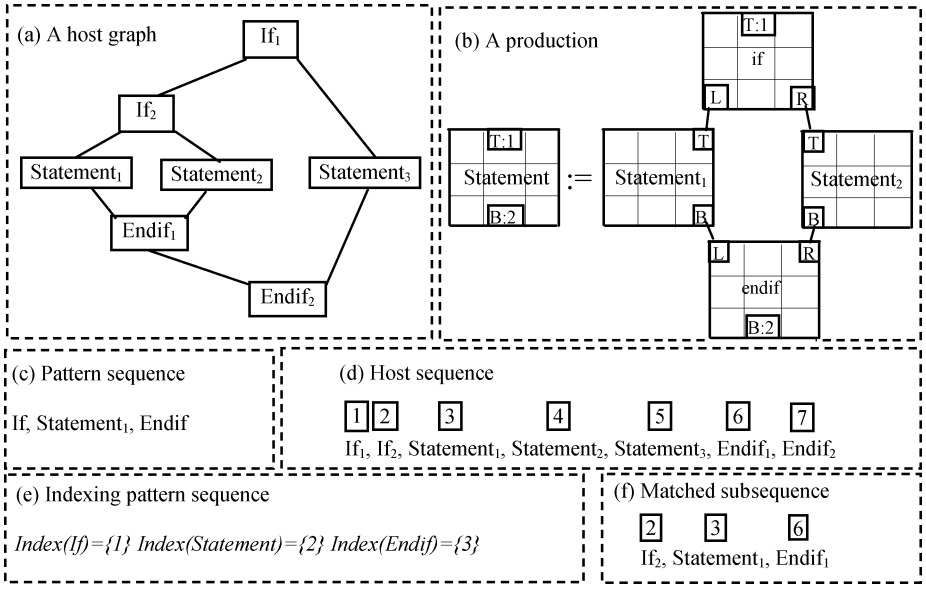


Fig. 16. Sequencing the host graph and production.

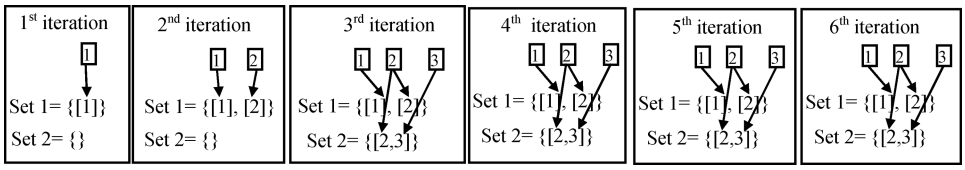


Fig. 17. A simulation of *FindRedexForR*.

5.2 An Example

The last section presents the matching algorithm in the SGG by ordering objects according to their spatial relationships. Based on an order among objects, we can narrow down the search space and thus perform an efficient matching. As an example, Figure 16(a) presents a host graph and Figure 16(b) illustrates a production where subscripts are used to distinguish objects of the same class. According to north-south relationships defined in the right graph, we can obtain two paths in the V-DAG, each containing three nodes. Similarly, two paths with three nodes can be derived from west-east relationships. We choose one of the four paths as the pattern sequence as shown in Figure 16(c). Correspondingly, the host sequence is demonstrated in Figure 16(d). The function *SequenceIndex* (see Figure 14) calculates the positions of a class of objects in the pattern sequence. In the previous example, the *If* class holds the first position, and thus *Index(If)* = {1}; the positions of the other two classes are shown in Figure 16(e).

Figure 17 traces the execution of *Match* for the example in Figure 16. Each object in the host sequence needs to be inspected until a redex has been found. Initially, all sets are empty. Since the first object of the host sequence can be mapped to the first object of the pattern sequence, a subsequence containing

only one object represented as [1] is obtained at the first iteration. Similarly, a subsequence containing the second object of the host sequence, which is mapped to the first object of the pattern sequence, is generated and inserted into Set 1 at the second iteration. At the third iteration, extending subsequences in Set 1 with the 3rd object of the host sequence can produce two subsequences, that is, [1,3] and [2,3]. Since a morphism exists between [2,3] and the first two objects of the pattern sequence, the subsequence [2,3] is inserted into Set 2. On the other hand, the subsequence [1,3] fails the structural verification: the production requires an edge between the first two objects of the pattern sequence, and such an edge does not exist between the first and third objects of the host sequence. Therefore, the subsequence [1,3] is discarded. At the fourth iteration, two subsequences [1,4] and [2,4] are generated by appending the fourth object to subsequences in Set 1. Since no graph morphism is found between the first two objects of the pattern sequence and the first and fourth objects of the host sequence, the subsequence [1,4] is excluded. The subsequence [2,4] is discarded since the verification of a spatial morphism has failed. At the fifth iteration, still no new subsequence is obtained. At the sixth iteration, a pattern sequence as shown in Figure 16(f), which meets both structural and spatial configurations, is found in the host sequence.

After finding the pattern sequence in the host graph, we need to search for nonpattern objects, for instance, the $statement_2$ node in the production of the example presented. The nonpattern object is defined locating south-east and north-east to the *if* and *endif* nodes, which are mapped to the second and sixth objects in the host sequence. Therefore, only the fourth object or the fifth object can be a match of the nonpattern object (another pattern object occupies the third position), which definitely reduces the search space. Since a morphism between the right graph of the production and the subsequence [2,3,4,6] of the host sequence exists, a redex is found.

5.3 Parsing Complexity

We now proceed to analyze the parsing complexity of the presented spatial graph grammar formalism.

In the host sequence of a host graph, a subsequence matching the pattern sequence of a production is called a *quasiredex*. If the nonpattern objects in the production find their matches in the host graph, the nonpattern objects in the host graph, together with the quasiredex, construct a redex. We first discuss the theoretical time complexity, and then the complexity of the parsing algorithm which is, in practice, faster than the theoretical speed.

THEOREM 1. *The time complexity of searching for a redex in SGGs is* $O(\frac{1}{m!}(m+n)^2|G|^{m+n})$ *where m and n are the maximal numbers of pattern and nonpattern objects in the right graphs of all the productions in a grammar.*

PROOF. With m pattern objects and n nonpattern objects of a production, there are $O(\frac{|G|^m}{m!})$ sequences for a quasiredex (the m-combination) and $O(|G|^n)$ sequences for nonpattern objects (the n-permutation). Each quasiredex requires $O(m^2+n^2+mn)$ time to identify a morphism. Therefore, the time of searching for a redex takes $O(\frac{1}{m!}(m^2+n^2+mn)|G|^{m+n})(\leq O(\frac{1}{m!}(m+n)^2|G|^{m+n}))$. □

In particular, we discuss three special cases:

(1) $m = 0$. No spatial information is specified in the productions and the SGG degenerates to the RGG. The time complexity is $O(n^2|G|^n)$, equal to that of the RGG [Zhang et al. 2001a].
(2) $n = 0$. No nonpattern object is specified in the productions, and the time complexity is $O(\frac{1}{m!}m^2|G|^m)$.
(3) $m \neq 0, n \neq 0$. The productions include both pattern and nonpattern objects, and the time complexity is $O(\frac{1}{m!}(m+n)^2|G|^{m+n})$ ($\leq O((m+n)^2|G|^{(m+n)})$.

THEOREM 2. *Given a host graph G with a grammar, the time complexity of* FindRedexForR *is* $O(m^2|G|^2 + k(m+n)^2|G|^n)$, *m and n are the maximal number of pattern and nonpattern objects in the right graphs of all the productions in the grammar, and* $(k-1)$ *is the number of quasiredexes processed before the first redex is found.*

PROOF. The function *SequenceRightGraph* converts the right graph of a production into a DAG and finds the longest path of the DAG to represent the pattern sequence. Since there are $O((m+n)^2)$ relationships (i.e., edges), *SequenceRightGraph* runs in $O((m+n)^2)$ time.

The function *SequenceHostGraph* generates a unique sequence from a host graph, and takes $O(|G|\lg|G|)$ to sort the y-coordinates and x-coordinates.

The function *match* searches for a redex in the host graph. It proceeds as follows: 1) look for a quasiredex in the host sequence; 2) then identify nonpattern objects in the host graph. The ith object in the host sequence is paired with each jth object ($j < i$). We check whether there exists a morphism between each of such pairs and its corresponding pair in the right graph. Since an object can be matched to m positions at most in the pattern sequence, the time complexity of verification is $O(mi) \times O(m)$, i.e., $O(m^2 i)$. Since there are $|G|$ objects, the time complexity of searching for a quasiredex is $O(\sum_{i=1}^{|G|} m^2 i) = O(m^2|G|^2)$. Once a quasiredex is found, we need to identify nonpattern objects in the host graph, which takes $O((n+m)^2)$ (the time of verifying whether there exists a morphism between objects in the right graph of a production and their occurrences in the host graph) by $O(|G|^n)$ (the number of sequences of nonpattern objects), that is, $O((m+n)^2|G|^n)$. Since there are totally $(k-1)$ quasiredexes before the first redex is found, the time complexity of the second step is $O(k(m+n)^2|G|^n)$. Therefore, the time complexity of *match* is $O(m^2|G|^2 + k(m+n)^2|G|^n)$.

Consequently, the time complexity of *FindRedexForR* is $O(m^2|G|^2 + k(m+n)^2|G|^n)$. □

It is easy to prove that the time complexity of *FindRedexForR* is equal to or less than $O(\frac{1}{m!}(m+n)^2|G|^{m+n})$.

Case 1. Every right graph in the productions of a graph grammar contains exactly one node, that is, $m + n = 1$. Since spatial information is specified between a pair of nodes, it follows that $m = 0$ and $n = 1$. The time complexity of *FindRedexForR* is $O(|G|)$, which is equal to the theoretical time complexity.

Case 2. A graph grammar contains at least one production, whose right graph includes two or more nodes, that is, $m + n \geq 2$. It is obvious that $k \leq \frac{|G|^m}{m!}$. Therefore, it is true that $(m^2|G|^2 + k(m+n)^2|G|^n) \leq \frac{1}{m!}(m+n)^2|G|^{m+n}$.

THEOREM 3. *The time complexity of the parsing algorithm for a graph G is* $O(m^2|G|^3 + k(m+n)^2|G|^{n+1})$*, m and n are the maximal number of pattern and nonpattern objects in the right graphs of all the productions, and* $(k-1)$ *is the number of quasiredexes processed before the first redex is found.*

PROOF. We have proven that the total time complexity of *FindRedexForR* is $O(m^2|G|^2 + k(m+n)^2|G|^n)$. $G \rightarrow^* A$ must finish within G.T(p) steps, where $G.T(p) = (2C)^p|G|$ (C represents the maximum possible number of nodes in any right graph of productions. Full proof for this step can be found in the proof for the RGG's parsing complexity [Zhang et al. 2001a]). The time complexity of the parsing algorithm is therefore $(2C)^p|G| * O(m^2|G|^2 + k(m+n)^2|G|^n) = O(m^2|G|^3 + k(m+n)^2|G|^{n+1})$. □

Therefore, due to the additional spatial information available to the parser, the parsing complexity of the spatial graph grammar is generally lower than that of its predecessor, i.e., the Reserved Graph Grammar.

6. AN APPLICATION IN ADAPTIVE WEB INTERFACES

We have defined a spatial graph grammar formalism (SGG). With the feature of integrated spatial and structural specifications, the SGG is distinguished from other graph grammar formalisms by explicitly introducing layout into the semantics of graph grammars. The SGG is suitable for applications that operate on graphical objects. This section demonstrates the use of an SGG to derive the semantic structure of Web pages, which can potentially impact many applications such as the adaptive Web interfaces discussed in detail in the following.

6.1 Background

The popularity of mobile devices, such as PDAs and cellular phones, has caused a dramatic increase in the diversity of computing devices in use [Myers et al. 2000]. Mobile devices provide a convenient and promising means to access the Web. Their usage for ubiquitous access to information is, however, seriously limited by their small screen size [Chen et al. 2003]. Interfaces on mobile devices cannot typically use the standard desktop model. Because most Web pages are tailored to personal computers and are not for displaying on mobile devices, users need to frequently scroll the window to find the content of interest. Due to the explosive growth of mobile devices, presenting an adaptive view on small screens has attracted much attention recently. Most researches [Chen et al. 2003; Yang and Zhang 2001] work on visual appearances of Web pages, while lacking a formal basis. This section introduces a visual approach based on the formal foundation of the spatial graph grammar formalism (SGG). Our approach uses the SGG to parse a Web page and generate a semantic structure. Such a semantic structure hierarchically reveals the composition among blocks

of information. Generally, a node at a higher level represents a bigger block of information which contains information represented by node(s) at a lower level. Based on the extracted semantic structure, we can split the Web page into several sub-pages, which consist of closely related information and are suitable for viewing on mobile devices. Detecting the semantic structure underlying a Web page is a technical challenge which is addressed by the application of the SGG.

Buyukkokten et al. [2001] divided a Web page into several semantic textual units with the help of HTML tags. For example, Tag *P* indicates the boundary between two semantic textual units. However, semantically related contents may be displayed close to each other, while they are far apart in the source HTML and cannot be grouped together by simply analyzing HTML tags. Therefore, layout also plays an important role in analyzing a Web page. In general, a good Web developer often observes some guidelines to render information on the Web with a comfortable layout, which allows users to catch information of interest easily. For example, related contents should be presented in a consistent style, and visual cues (e.g., lines and blanks) are used to separate contents of different topics. These ad hoc guidelines motivate the heuristic approaches [Chen et al. 2003; Yang and Zhang 2001; Yu et al. 2003] based on the analysis of the visual appearance of a Web page. Those approaches are automatic and efficient in grouping relevant information but cannot recover the semantic roles of information blocks from their layout structures. Different from heuristic approaches, the SGG offers a formal approach which takes both the HTML structure and visual appearance of a Web page into account. Therefore, our approach not only utilizes the hints hidden under the HTML structure, but also groups elements which are spatially close when displayed, but far apart in the source HTML. The grouping is realized through spatial parsing.

Recently, visual language formalisms have been applied to analyzing patterns of Web queries [Zhang et al. 2004]. Given a grammar in the form of a variant of the attributed multiset grammar [Golin 1991], which specifies commonly used Web query patterns, a best-effort parser analyzes a query form by parsing the spatial arrangement of visual objects inside the form. Without considering the HTML DOM structure, it is possible for the approach to falsely combine two visual objects crossing a block-level HTML element (e.g., *TR*). Such errors can be avoided in our approach by considering both HTML DOM structure and visual appearances of a Web page.

6.2 Adaptive Web Interfaces

A Web developer often has a high-level semantic structure in mind when creating a Web page. Though such a structure usually disappears after the Web page is constructed and displayed to the user [Chen et al. 2003], cues about the semantic structure can still be found in the HTML DOM structure and from its visual appearance. For example, semantically related information is often enclosed in a common block-level HTML tag (e.g., *TABLE*), and a topic name is generally placed on top of the detailed contents. Based on the above observations, our visual approach to adaptive viewing takes the HTML DOM structure with the visual appearance of a Web page as input, which is then parsed by the SGG parser to generate a semantic structure. The generated

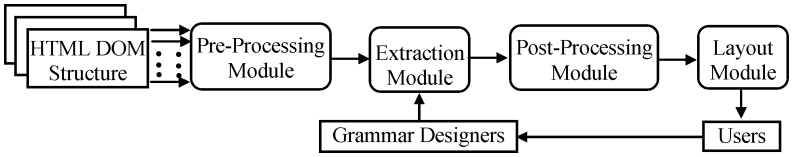


Fig. 18. A visual approach to adaptive viewing on mobile devices.

semantic structure will be extremely useful for meaningful content grouping to suit different presentation environments. The following example demonstrates the generation of a semantic structure to suit small screen presentations.

Figure 18 presents our approach in which the extraction module is the kernel. We go through an example to present a complete picture of an adaptive Web interface based on the spatial graph grammar formalism and explain the role of each module in Figure 18.

Consider an example that a museum exhibits its collections online (www.moma.org). The museum classifies its collections into several categories maintained by different departments. All departments have a large number of pages using a common template. Figure 19 illustrates one of the pages from the department of painting and sculpture. The page consists of a navigation bar with various hyperlinks on the left side, hyperlinks to all departments in the middle of the right side, and the painting collections occupying the main area. The underlying semantic structure, which will be extracted through our approach, is briefly annotated in Figure 19. Figure 20(a) gives a fragment of HTML source codes whose corresponding HTML DOM structure is presented in Figure 20(b) for the Web page in Figure 19. Such a DOM structure, which includes the spatial information of each node (such as positions) used for spatial parsing, serves as the input of our approach.

Since some HTML tags, such as *TABLE* and *P*, are used not only for layout, but also for content organization [Yu et al. 2003], the HTML DOM structure provides valuable cues for recovering the underlying semantic structure. Among the ninety-one tags in HTML 4.0, many of them are just for layout purpose.Taking all the tags into consideration would be disastrous for grammar developers and cause inefficient parsing. The preprocessing module is designed to generate a simplified HTML DOM structure by keeping content organization-related tags (such as *TABLE* and *TR*) and removing all other tags. Using the HTML DOM structure in Figure 20(b), Figure 21 gives a simplified HTML DOM structure after preprocessing in which the leaf nodes, such as *text* and *picture*, represent single Web objects displayed on the screen.

The extraction module is the key to the whole approach for recovering the underlying semantic structure which reveals the composition of Web objects. The extraction process is fundamentally a parsing process guided by a graph grammar. A derivation sequence, which records the history of production applications, implies the information organization. Keeping the previous example in mind, Figure 22 shows the graph grammar that recovers the semantic structure for a large number of Web pages using a common museum template. Figure 23

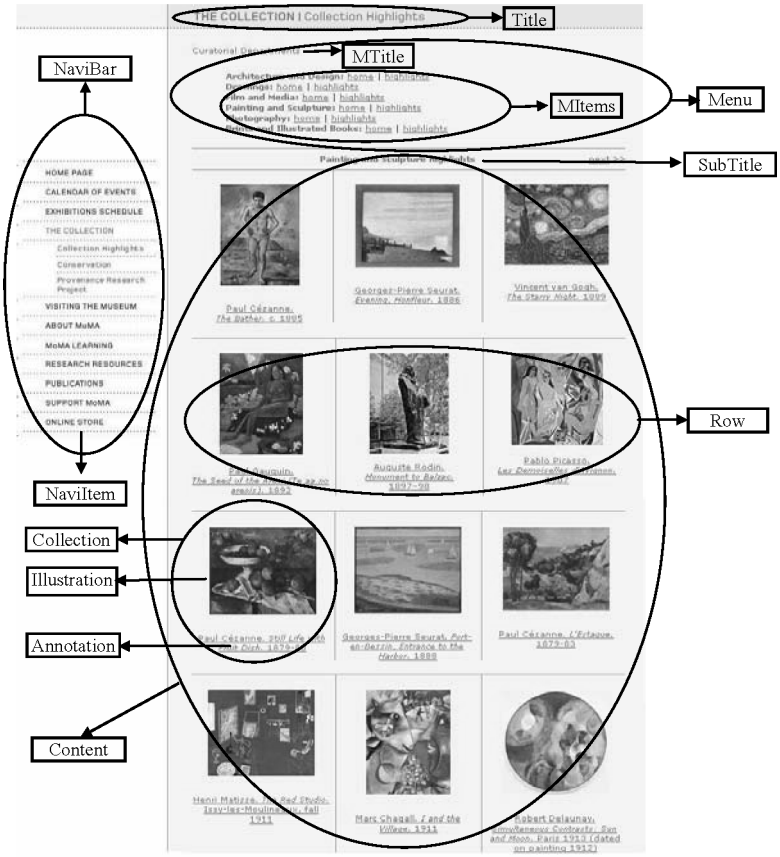
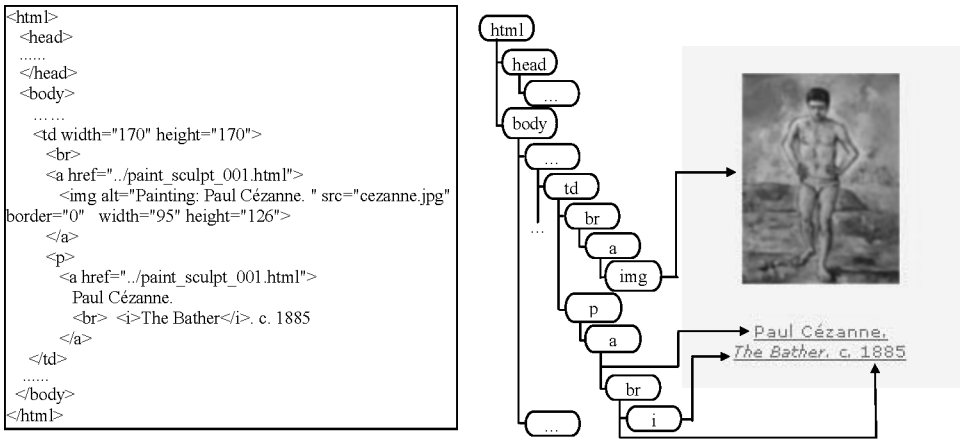


Fig. 19. A Web page.

presents the corresponding action codes[11] associated with productions. Production <1> specifies that the Web page consists of four composite Web objects, that is, *Title, NaviBar (Navigation Bar), Menu, and Content*. The *Title* is placed on the top of the page. The *NaviBar* is displayed on the left side, while the *Menu* and *Content*, which have a north-south relation, are shown on the right side. The dotted lines in productions specify the spatial relationships between objects without an HTML structural relationship. Production <2> abstracts a single Web object *Text*, which is enclosed in a *P* tag inside the *BODY* tag, to a composite object *Title*. Productions 3, 4, and 5 reduce a list of pictures inside a common *TABLE* tag to a composite object *NaviBar*. Productions 6 and 7 define that a *Collection* object is made up of a *Picture* and a *Text* enclosed in the same *TD* tag. Production 8 reduces *Collection* objects inside a *TABLE* tag to a *Row*

[11]The function *CopySpatialProperty (Object* O_1*, Object* O_2*)* copies the spatial information, such as position, from the source object O_2 to the destination object O_1. The function *Merge(Object* O_1*, Object* O_2*, Object* O_3*)* sets the spatial attributes of the Oject O_1 with the minimal bounding box covering both O_2 and O_3.



(a) A fragment of an HTML document

(b) An HTML DOM structure

Fig. 20. A Fragment of an HTML document and its HTML DOM structure.

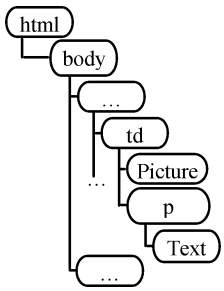


Fig. 21. A simplified HTML DOM structure.

object. Production 9 defines the abstraction of a *SubTitle* object from a single Web object *Text*. Productions 10 and 11 illustrate the composition of a *Content* object. Productions 12, 13, and 14 present the construction of *Menu* object.

Applying the graph grammar to the museum Web pages, a sequence of production applications derives the semantic structure among Web objects. For example, Figure 24(a) gives a host graph where a redex to the right graph of Production 6 is enclosed in a dotted rectangle. The application of Production 6 indicates that objects *Illustration* and *Annotation* are composed of single Web objects *Picture* and *Text*, respectively. We can further apply Production 7 to the graph in Figure 24(b) to derive the composite object *Collection* from HTML elements *TD* and *P* and objects *Illustration* and *Annotation*.

The extraction module generates a derivation sequence which records the application history of productions and implies the semantic structure of a Web page. The derivation sequence not only contains Web objects but also HTML tags. The postprocessing prunes the derivation sequence and generates a clean semantic structure between Web objects by removing HTML tags, ignoring context information of production application, and removing intermediate nodes. For example, the *Collection* object is simply composed of objects *Illustration* and

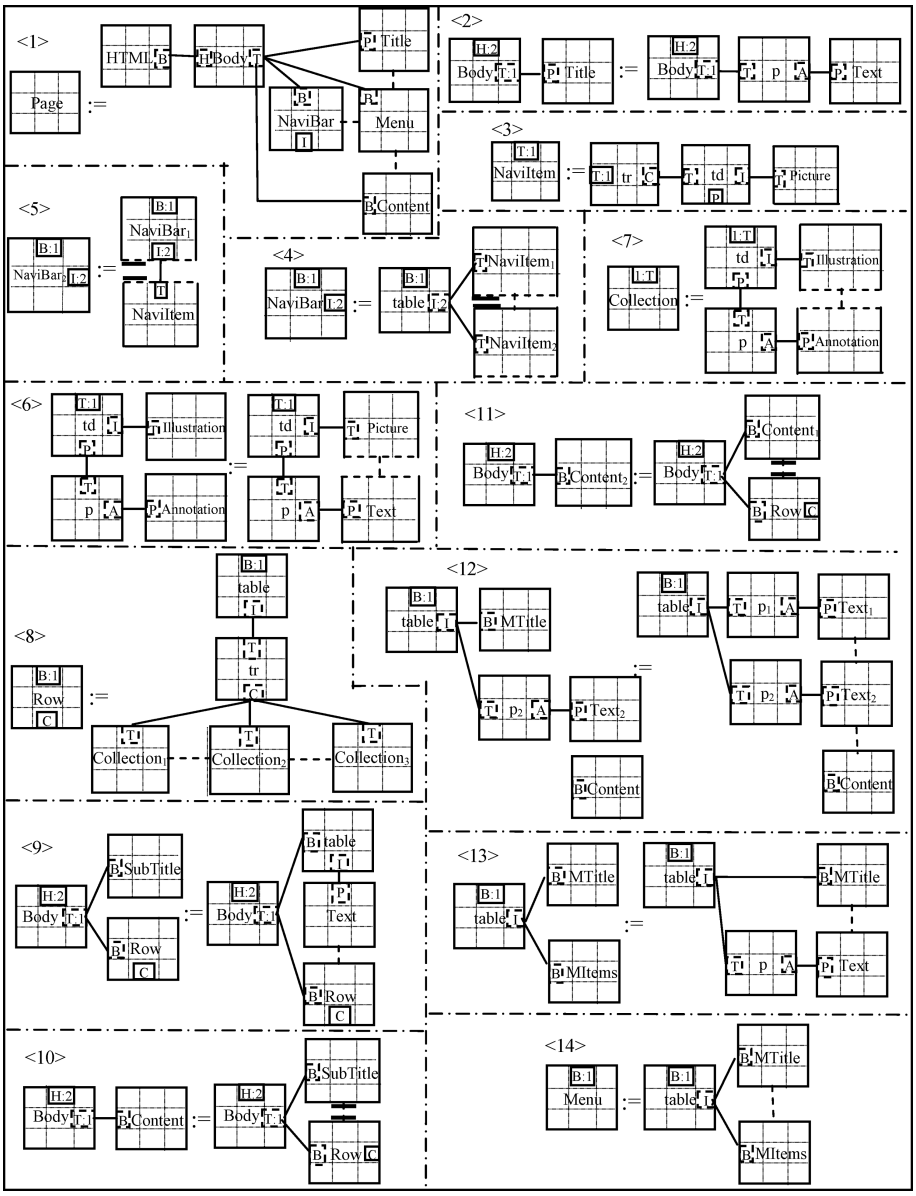


Fig. 22. A graph grammar extracting semantic structure.

Annotation when we remove the HTML tags. After the postprocessing, a final semantic structure for the Web page in Figure 19 is given in Figure 25.

The extracted semantic structure is extremely useful in generating adaptive layout on mobile devices. Based on the extracted semantic structure, related information, which may not even be displayed adjacent to the original Web page, can be flexibly grouped together to generate a subpage appropriate for displaying on a small screen of mobile devices. For example, the following algorithm

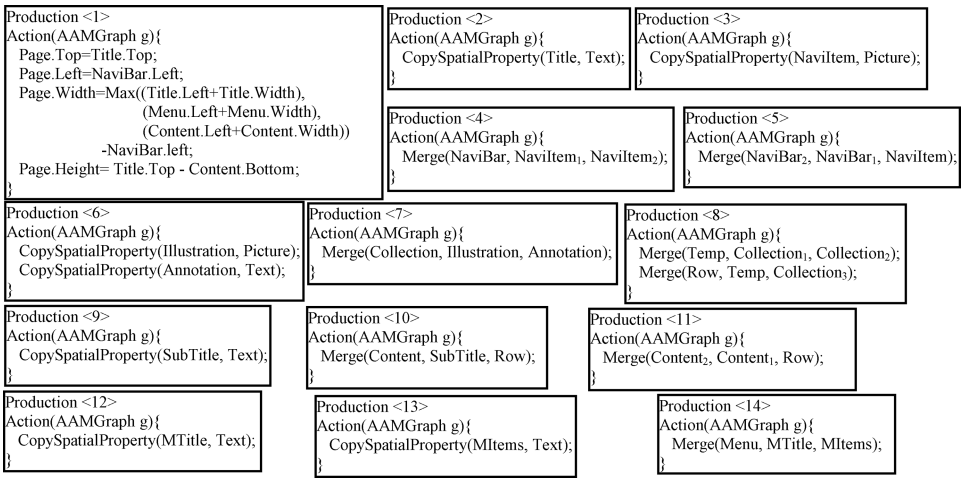


Fig. 23. Action codes for the graph grammar in Figure 22.

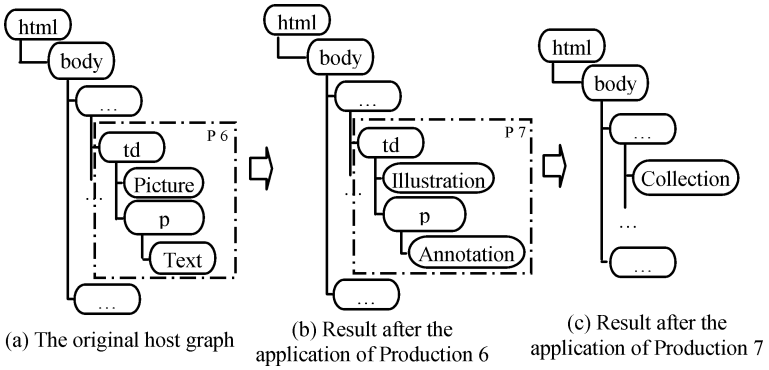


Fig. 24. A sequence of production application.

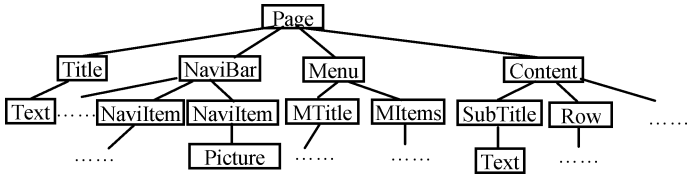


Fig. 25. The hierarchical structure for the Web page presented in Figure 21.

(Figure 26) can attach an appropriate title to a subpage and use thumbnails as content indices to support incremental zooming. Alternatively, depending on the size and browsing capability, we can adapt the algorithm to present contents in a textual representation which is more suitable for displaying on cellular phones. Therefore, based on the extracted structure, various layout styles can be generated for corresponding mobile devices.

We will generate two types of subpages, that is, *index pages* and *displaying pages*. An index page serves as an index of the contents and is displayed

```
Splitting(Node root)
{
  Push root into a stack;
  While (stack is not empty) {
    currentNode ← Popup the stack;
    if the currentNode is not a heading element{
      Get current viewing environment
      if the size of currentNode fits on the viewing environment
        generate a displaying page;
      else {
        generate an index page;
        push its child nodes to the stack;
      }
    }
  }
}
```

Fig. 26. Splitting algorithm.

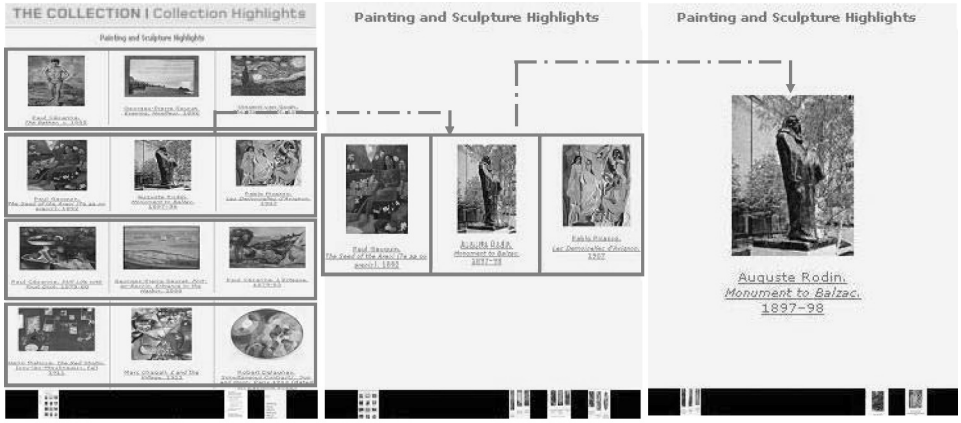
as a thumbnail, in which users can click on a block directing them to a corresponding subpage with an appropriate content. A displaying page presents information on mobile devices with a pleasant layout which allows users to read more details. Each subpage is made up of three parts:

—an appropriate title is placed on the top;
—contents are presented in the main area; and
—a list of icons linking to relevant subpages are displayed at the bottom.

In the extracted structure, nodes with certain types are distinguished as *heading elements*. A heading element n_1 will not generate a separate subpage. Instead, it is displayed as a heading in subpages. In the extracted structure in Figure 25, nodes of types *Title*, *SubTitle*, and *MTitle* are heading elements. *Title* serves as the heading for subpages generated from nodes *NaviBar*, *Menu*, and *Content*, while *SubTitle* serves as the heading for subpages from *Row*. In order to ease browsing, each subpage includes a list of small navigation icons linking to the subpages of its sibling and parent nodes.

The splitting process starts from the root node of the extracted semantic structure. If the size of the currently processed node fits on the current viewing environment, a subpage is generated for displaying; otherwise, we generate an index subpage and continue to process its child nodes. The procedure proceeds as in Figure 26.

With the extracted structure in Figure 25, the root node (i.e., *Page*) generates an *index page* with three blocks which link to the subpages generated from its three child nodes (*Page* has four child nodes but the *Title* node is a heading element which does not generate a subpage). Since *Page* is the root without any sibling and parent node, there is no navigation icon in the generated index page. Then, we proceed to the child nodes of *Page*. Nodes *NaviBar* and *Menu* correspondingly generate two displaying pages, while node *Content* generates an index page due to its large size. Figure 27(a), which is manually produced according to the parsing result (since the layout part is under development), presents the subpage of node *Content*. A title is displayed on the top. The main



(a) A *Content* subpage (b) A *row* subpage (c) A collection subpage

Fig. 27. Page generation based on the extracted structure.

area includes four information blocks linking to the subpages of its child nodes. The icon on the bottom left links to the subpage of its parent node, and the two icons on the bottom right link to subpages of its sibling nodes. When clicking on an information block, the user can go into a corresponding subpage as shown in Figure 27(b) with zoom-in details in Figure 27(c).

In summary, this section illustrates the application of the spatial graph grammar formalism to adaptive Web interfaces. Unlike other works [Borning et al. 2000; Marriott et al. 2002; Zhang et al. 2005] focusing on designing new Web pages with adaptive capability, we extend existing Web pages to support adaptive layouts based on the spatial graph grammar formalism. To the authors' best knowledge, our approach represents the first attempt at applying graph grammars to deriving the semantic structure of Web pages in the context of adaptive Web interfaces.

7. OTHER APPLICATIONS

The previous section demonstrated a scenario about adaptive Web interfaces using the spatial graph grammar formalism. This section discusses some other applications.

A straightforward application of the SGG is the automatic generation of diagram editors. A diagram editor is tailored to a specific diagram language and restricted to visual objects which occur in the diagram language [Minas 2002]. Diagram editors have advantages over pure drawing tools by recognizing the edited diagrams. Two standard interaction modes for diagram editors are freehand editing and syntax-directed modes. A freehand editing mode allows the end-users to manipulate the diagram freely during diagram creation. A parser is necessary to analyze the syntactic structure of the created diagram. This mode does not automatically preserve the semantics of the diagram during manipulation. On the other hand, the syntax-directed mode guides the editing by providing appropriate operations during the diagram construction and thus enforces a specific order to construct a diagram. This mode does not allow

incorrect syntactic structure for intermediate diagrams as the editing process is guided by a sequence of structural templates [Chok and Marriott 2003].

Another application area is graph drawing [Battista et al. 1999]. The potential applications in graph representation give rise to the research in graph drawing. The main purpose of graph drawing is to develop an efficient layout algorithm that can generate a readable and pleasant-looking graph representation. In different fields, the layout of a graph needs to meet different aesthetic rules. A layout algorithm has to calculate the geometrical positions of graph elements carefully according to the whole structure of the graph. A graph grammar introduces a formal framework to generate a syntax-directed layout by capturing the internal structure. The spatial graph grammar formalism is applicable to graph layout.

—Provided that a host graph is well-formed in terms of abstract structures, a hierarchical graph, which denotes a derivation history on the host graph, is obtained through a parser. Then a syntax-directed layout is directed based on the hierarchical graph. Therefore, a host graph in the application of graph drawing only denotes a pure structural configuration without any spatial information.

—The spatial relationships defined in the productions represent the desirable layout between constructs in the host graph. In other words, spatial specifications embedded in the graph grammar form a set of syntax-directed geometrical constraints for corresponding pictorial objects in the host graph.

In summary, the spatial graph grammar formalism serves as both the visual programming language grammar and the layout grammar. However, describing the routing of edges with graph grammars is still an open problem [Brandenburg 1995].

The spatial graph grammar formalism is suitable for semantic analysis. An effective application of the combined use of abstract and spatial specifications is the spreadsheet paradigm of visual programming languages [Burnett and Gottfried 1998; Burnett et al. 2002]. The position of a spreadsheet cell can determine the semantics of the cell. In many applications, an order over a set of objects is required to satisfy some semantic constraints. Since an order between two objects can be visually represented in a two-dimensional graph, it is natural to encode an order using spatial relationships, and the parser can verify a predefined sequence over a set of objects.

8. RELATED WORK

To the authors' knowledge, research on combining both spatial and abstract specifications in graph grammars has been scarce. The SGG differs from other graph grammars by introducing spatial information to the abstract syntax. Layout does not have any meaning in other graph grammars, which is changed by our approach, in which layout is explicitly represented. The direct representation of spatial information in productions can make the productions easier to understand since developers often design rules in a manner similar to their specified diagrams. Furthermore, the SGG is equipped with a parser which is

more efficient than its predecessor in using spatial information. The following description discusses other approaches and compares them with the SGG.

Lakin [1987] presented the concept of spatial parsing and described a type of grammar for specifying visual programming languages. In particular, the proposed grammatical approach allows the specification of a spatial arrangement among symbols on the right-hand side. However, the annotated grammars are not formalized.

Using *multisets* (i.e., unordered collections) as the underlying data model, the formalism of *Picture Layout Grammars* (PLGs) [Golin and Reiss 1989] has been proposed to define pictures, which are viewed as unordered collections of visual symbols with attributes containing positional information. Improving on the PLG, *Constraint Multiset Grammar* (CMG) [Marriott 1994] is a high-level framework for the definition of visual programming languages. A production in a constraint multiset grammar is in a form.

$$\mathrm{P} ::= \mathrm{P}_1, \ldots, \mathrm{P}_n, \text{ where exist } \mathrm{P'}_1, \ldots, \mathrm{P'}_m \text{ where } C$$

This production indicates that the nonterminal symbol P can be rewritten by the multiset of symbols $\mathrm{P}_1,\ldots,\mathrm{P}_n$ whenever there exist $\mathrm{P'}_1,\ldots,\mathrm{P'}_m$ such that the attributes of all symbols satisfy the constraint C. As a high-level grammatical specification language, the CMG supports the generation of diagram editors in the Penguins system [Chok and Marriott 2003]. Taking the specification of a diagram language as the input, the system automatically generates an incremental diagram parser which interprets diagrams according to the relationships between objects captured by geometrical constraints. In particular, such a diagram editor combines the best features of the freehand and syntax-directed modes.

The *Relational Grammar* extends traditional one-dimensional string languages to higher dimensions through user-supplied domain relations [Weitzman and Wittenburg 1993]. In other words, sentential forms are specified as multisets of symbols with a set of relations in the symbol set [Wittenburg 1992]. Both the CMG and the Relational Grammar are characterized by generating or parsing languages according to relational constraints among objects in a textual form.

The three formalisms just discussed fall into the class of multiset rewriting. Unlike graph grammars that enforce a strict distinction between objects (nodes) and relations (edges) [Bardohl et al. 1999], they manipulate all needed spatial or abstract relationships implicitly through constraints over attribute values of objects. Furthermore, abstract syntax graphs do not exist as a distinct notion within their syntax definitions.

Rekers and Schürr [1996] classified the spatial relations graph (SRG) and abstract syntax graph (ASR). The former is geared towards visualization and the latter towards interpretation. A triple graph grammar [Schürr 1994], that is, the coupled graph grammar, is used to build up interdependencies between SRGs and ASRs. The coupled graph grammar is used to define a diagram editor [Rekers and Schürr 1996]. The editor updates the layout of a diagram on the basis of changes in the SRG. Since multiple layouts may exist to satisfy spatial relationships in a SRG, Rekers and Schürr proposed *layout editing*, which

allows users to change the layout with the updated layout still satisfying all spatial constraints.

In the coupled graph grammar, spatial relationships are defined over pictorial objects, while in the SGG, these are specified over abstract objects each of which is represented by one or several pictorial objects. Consequently, spatial configurations are specified at a higher level in the SGG than in the coupled graph grammar. In general, an SGG specification supplemented by internal layouts of abstract objects identifies the same information as a coupled graph grammar. Another fundamental difference is that the SGG integrates spatial and abstract specifications within a single grammar instead of a pair of grammars, and thus the SGG introduces a new concept to high-level specification languages by supporting definitions of relationships with both edges and spatial constraints.

DIAGEN is a prototyping tool for creating diagram editors which uses hypergraphs rather than box-and-line graphs as in the SGG, to model various types of diagrams [Minas 2002]. In the hypergraph model, several distinct *attachment areas* of a diagram component connect to other diagram components and establish spatial relationships. In particular, edges in the model are distinguished as *component edges* and *relation edges*. The former represents diagram components, and the latter indicates relationships between attachment areas.

Bottoni et al. [1999] introduced the Visual Conditional Attributed Rewriting (VCARW) system to specify visual interactive systems based on direct manipulation. In an interactive session, images denoting exchanged messages are materialized and interpreted by the human and the computer. The human interprets an image by recognizing characteristic structures. On the other hand, the computer uses a set of attributed symbols to capture the meaning of an image. Necessary spatial information, such as the physical position of an object, is identified through corresponding symbols but does not participate in spatial transformation.

Brandenburg [1995] presented a layout graph grammar consisting of an underlying context-free graph grammar and layout specifications. Spatial relationships are derived according to the drawing of productions. A desirable layout is achieved by satisfying these constraints. One serious drawback of the approach is that grids and planar graphs cannot be captured by context-free graph grammars [Brandenburg 1995]. We described a grammar-based approach for graph drawing [Zhang and Zhang 2002] in which layout rules are embedded in the productions of a context-sensitive graph grammar formalism. The spatial relationship is represented by labels. This approach is suitable for graph drawing of visual programs. These two formalisms only focus on the graph layout.

Of three different approaches to visual languages specification [Marriott et al. 1998], the logical approach uses mathematic logic to axiomatize the possible spatial relationships between objects. Meyer [1992] proposed a visual logic formalism, that is, the *Picture Clause Grammar* (PCG) which integrates logic programming with graphical expressions. The PCG introduces visual terms into logic programming, that is, *picture term* partially describing the contents of a picture, and is distinguished by the combination of spatial reasoning with the expressiveness of graphical communications. Another prominent logic

approach is based on *spatial logic* [Haarslev 1998] which is made up of three components, namely, geometric objects, spatial relations, and description logic [Brachman and Schmolze 1985]. Capable of describing syntax and semantics of visual language in a uniform framework, the spatial logic serves as the theory underlying an object oriented editor, specifically GenEd [Haarslev and Wessel 1996], which can define visual languages such as Petri nets etc. Falling into the category of grammatical approaches, our work has a different theoretical root from these logic approaches.

Our work is also related to image retrieval using symbolic descriptions. Due to the large amount of information associated with images, efficient retrieval needs suitable representation of spatial information to index image contents. Chang et al. [1987] proposed 2D symbolic strings to encode spatial relationships between objects as projected along the two coordinate axes. Del Bimbo et al. [1994] presented a logic-based language for describing the contents of stored images and specifying queries. The language specializes in the syntax and the semantics of *temporal logic* to deal with spatial ordering between the projections of objects in a scene. Soffer and Samet [1998] presented a pictorial query specification technique for image databases and addressed the issues of matching contextual and spatial ambiguity inherent in pictorial queries. Each query is composed of one or more query images by selecting required objects and positioning them according to the desired spatial configuration. These image retrieval techniques have inspired us to develop a precise semantics for spatial relationships within a grammatical framework.

9. CONCLUSION

Physical layout and abstract structure are two aspects of a graph. This article has presented a spatial graph grammar formalism which serves both abstract syntax grammar and layout grammar. As a context-sensitive graph grammar formalism, it is equipped with a parser that performs in polynomial time with an improved parsing complexity over its nonspatial predecessor, the Reserved Graph Grammar. The node-edge representation is very similar to box-and-line drawings which meet the common practice of software engineers [Métayer 1998]. With the capability of specifying spatial and structural information in a uniform fashion, the SGG is suitable as the underlying grammar for various prototyping tools such as adaptive representations, diagram editors, Web information transformation, etc.

The SGG treats spatial relationships as language constructs. Therefore, a spatial arrangement can affect the interpretation of a visual sentence. The SGG can derive the spatial signature of a host graph according to the positions of objects (a quantitative representation of spatial information among objects) in a planar space. The SGG parser verifies such a spatial signature, that is, a qualitative representation capturing the spatial properties among objects, against the spatial specifications in a grammar. A graph is a valid visual sentence only if it satisfies both the structural and spatial specifications defined in a spatial graph grammar.

In the SGG, the application of a production can rearrange the spatial configuration of a host graph almost arbitrarily and may cause inconsistent spatial

relationships. In order to ensure the integrity of a sequence of transformations, we need to guarantee a consistent set of spatial specifications defined in a graph grammar. We recently proposed a graph model to detect directional inconsistency [Kong and Zhang 2003]. However, detecting inconsistency, when incorporating the other three spatial aspects into the graph model, raises challenging issues which deserve further investigation.

Writing productions and their action codes to perform visual computing is not an easy task even for a design expert since it requires a good command of the graph grammar formalism. It has been the authors' goal to partially automate the production authoring from application constraints.

ACKNOWLEDGMENTS

The authors would like to thank the Associate Editor Brad A. Myers and the anonymous reviewers for their insightful and constructive comments that have helped us to significantly improve the presentation.

REFERENCES

ALLEN, J. 1983. Maintaining knowledge about temporal intervals. *Comm. ACM* 26, 832–843.

ALLEN, R. AND GARLAN, D. 1994. Formalizing architectural connection. In *Proceedings of the 16th International Conference on Software Engineering*. 71–80.

BARDOHL, R., TAENTZER, G., MINAS, M., AND SCHÜRR, A. 1999. Application of graph transformation to visual languages. In *Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*. H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, Eds. World Scientific, 105–180.

BATTISTA, G. D., EADES, P., TAMASSIA, R., AND TOLLIS, I. G. 1999. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, Englewood Cliffs, NJ.

BLOSETIN, D. AND SCHÜRR, A. 1999. Computing with graphs and graph transformation. *Softw. Practice Exper. 29*, 3, 197–217.

BORNING, A., LIN, R. K., AND MARRIOTT, K. 2000. Constraint-based document layout for the web. *Multimedia syst. 8*, 3, 177–189.

BOTTONI, P., COSTABILE, M. F., AND MUSSIO, P. 1999. Specification and dialogue control of visual interaction through visual rewriting systems. *ACM Trans. Programm. Lang. Syst. 21*, 6, 1077–1136.

BOTTONI, P. AND MINAS, M. Eds. 2003. *Proceedings of the GT-VMT'2002-Graph Transformation and Visual Modeling Techniques*, Electronic Notes in Theoretical Computer Science *72*, 3. Elsevier Science.

BRACHMAN, R. J. AND SCHMOLZE, J. G. 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 171–216.

BRANDENBURG, F. J. 1995. Designing graph drawings by layout graph grammars. In *Proceedings of the International Workshop on Graph Drawing*. 416–428.

BURNETT, M. M. AND GOTTFRIED, H. J. 1998. Graphical definitions: Expanding spreadsheet languages through direct manipulation and gestures. *ACM Trans. Comput.-Human Interact. 5*, 1, 1–33.

BURNETT, M. M., YANG, S., AND SUMMET, J. 2002. A scalable method for deductive generation in the spreadsheet paradigm. *ACM Trans. Comput.-Human Interact. 9*, 4, 253–284.

BURNETT, M. M. 2006. Visual language research bibliography. http://www.cs.orst.edu/~burnett/vpl.html.

BUYUKKOKTEN, O., GARCIA-MOLINA, H., AND PAEPCKE, A. 2001. Accordion summarization for end-game browsing on PDAs and cellular phones. In *Proceedings of the ACM SIGCHI'01*. 213–220.

CHANG, S. K., SHI, Q. Y., AND YAN, C. W. 1987. Iconic indexing by 2-D string. *IEEE Trans. Patt. Analy. Machine Intelli. 9*, 413–427.

CHEN, Y., MA, W. Y., AND ZHANG, H. J. 2003. Detecting web page structure for adaptive viewing on small form factor devices. In *Proceedings of the International Conference on World Wide Web*, 225–233.

CHOK S. S. AND MARRIOTT, K. 2003. Automatic generation of intelligent diagram editors. *ACM Trans. Comput.-Human Interact. 10*, 3, 244–276.

CLEMENTINI, E., FELICE, P. D., AND OOSTEROM, P. V. 1993. A small set of formal topological relationships for end-user interaction. In *Proceedings of the 3rd International Symposium on Advances in Spatial Databases*. Lecture Notes in Computer Science, vol. 692, 277–295.

CLEMENTINI, E., FELICE, P. D., AND HERNÁNDEZ, D. 1997. Qualitative representation of positional information. *Artificial Intelli. 95*, 317–356.

COHN, A. G. 1997. Qualitative spatial representation and reasoning techniques. In *Proceedings of the KI-97*. Lecture Notes in Artificial Intelligence, vol. 1303, 1–30.

COHN, A. G. AND HAZARIKA, S. M. 2001. Qualitative spatial representation and reasoning: An overview. *Fundamenta Informaticae 46*, 1–2, 1–29.

DEAN, T. R. AND CORDY, J. R. 1995. A syntactic theory of software architecture. *IEEE Trans. Soft. Engin. 21*, 4, 302–313.

DEL BIMBO, A., VICARIO, E., AND ZINGONI, D. 1994. A spatial logic for symbolic description of image contents. *J. Visual Lang. Comput. 5*, 3, 267–286.

EHRIG, H., ENGELS, G., KREOWSKI, H. J., AND ROZENBERG, G. Eds. 1999a. *Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*. World Scientific.

EHRIG, H., KREOWSKI, H. J., MONTANARI, U., AND ROZENBERG G. Eds. 1999b. *Handbook of Graph Grammars and Computing by Graph Transformation: Concurrency, Parallelism, and Distribution*. World Scientific.

FISCHER, T., NIERE, J., TORUNSKI, L., AND ZÜNDORF, A. 1998. Story diagrams: A new graph rewrite language based on the unified modeling language and java. In *Proceedings of the Theory and Application to Graph Transformations*. Lecture Notes in Computer Science, vol. 1764, 296–309.

FRANK, A. U. 1996. Qualitative spatial reasoning: cardinal directions as an example. *Int. J. Geograph. Inform. Sci. 10*, 3, 269–290.

GOLIN, E. J. AND REISS, S. P. 1989. The specification of visual language syntax. In *Proceedings of the IEEE Workshop on Visual Languages*. 105–110.

GOLIN, E. J. 1991. Parsing visual languages with picture layout grammars. *J. Visual Lang. Comput. 4*, 2, 371–394.

HAARSLEV, V. AND WESSEL, M. 1996. GenEd–an editor with generic semantics for formal reasoning about visual notations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*. 204–211.

HAARSLEV, V. 1998. A fully formalized theory for describing visual notations. In *Visual Language Theory*. K. Marriott and B. Meyer, Eds. Springer, 261–292.

HECKEL, R., MENS, T., AND WERMELINGER, M. Eds. 2003. In *Proceedings of the Workshop on Software Evolution Through Transformations Toward Uniform Support Throughout the Software Lift-Cycle*. Electronic Notes in Theoretical Computer Science *72*, 4. Elsevier Science.

HERNÁNDEZ, D., CLEMENTINI, E., AND FELICE, P. D. 1995. Qualitative distance. In *Proceedings of the Spatial Information Theory: A Theoretical Basis for GIS*, Lecture Notes in Computer Science, vol. 988, 45–58.

IRANI, P. AND WARE, C. 2003. Diagramming information structures using 3D perceptual primitives. *ACM Trans. Comput.-Human Interac. 10*, 1, 1–19.

KONG, J. AND ZHANG, K. 2003. Graph-based consistency checking in spatial information system. In *Proceedings of the 2003 IEEE Symposium on Visual Languages and Formal Methods*. 153–160.

KREOWSKI, H. J. AND KNIRSCH, P. Eds. 2002. In *Proceedings of the AGT2002-Applied Graph Transformation*.

LAKIN, F. 1987. Visual grammars for visual languages. In *Proceedings of the American Association for Artificial Intelligence*. 683–688.

MARRIOTT, K. 1994. Constraint multiset grammars. In *Proceedings of the IEEE Symposium on Visual Languages*. 118–125.

MARRIOTT, K. AND MEYER, B. 1997. On the classification of visual languages by grammar hierarchies. *J. Visual Language Comput. 8*, 375–402.

MARRIOTT, K., MEYER, B., AND WITTENBURG, K. B. 1998. A survey of visual languages specification and recognition. In *Visual Language Theory*. K. Marriott and B. Meyer, Eds. Springer, 5–85.

MARRIOTT, K., MEYER, B., AND TARDIF, L. 2002. Fast and efficient client-side adaptability for SVG. In *Proceedings of the World Wide Conference*. 496–507.

MENS, T., SCHÜRR, A., AND TAENTZER, G. Eds. 2002. In *Proceedings of the GraBaTs 2002-Graph-Based Tools*. Electronic Notes in Theoretical Computer Science *72*, 2, Elsevier Science.

MÉTAYER, D. L. 1998. Describing software architecture styles using graph grammars. *IEEE Trans. Soft. Engin.* 24, 7, 521–533.

MEYER, B. 1992. Pictures depicting pictures on the specification of visual languages by visual grammars. In *Proceedings of the IEEE Workshop on Visual Languages*. 41–47.

MINAS, M. 2002. Concepts and realization of a diagram editor generator based on hypergraph transformation. *Science of Computer Program*. 40, 157–180.

MYERS, B., HUDSON, S. E., AND PAUSCH, R. 2000. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Human Interact. 7*, 1, 3–28.

PEUQUET, D. AND ZHAN, C. X. 1987. An algorithm to determine the directional relationship between arbitrarily-shaped polygons in a plane. *Patt. Recog.* 20, 65–74.

PULLAR, D. AND EGENHOFER, M. 1988. Toward formal definitions of topological relationships. In *Proceedings of the 3rd International Symposium on Spatial Data Handling*. 225–241.

QIU, M. K., SONG, G. L., KONG, J., AND ZHANG, K. 2003. Spatial graph grammars for Web information transformation. In *Proceedings of the IEEE Symposium on Visual/Multimedia Languages*. 84–91.

REKERS, J. AND SCHÜRR, A. 1996. A graph based framework for the implementation of visual environments. In *Proceedings of the IEEE Symposium on Visual Languages*. 148–155.

REKERS, J. AND SCHÜRR, A. 1997. Defining and parsing visual languages with layered graph grammars. *J. Visual Languag. Comput. 8*, 1, 27–55.

RÉVÉSZ, G. E. 1983. *Introduction to Formal Languages*. McGraw-Hill Inc.

ROZENBERG, G. AND WELZL, E. 1986. Boundary NLC graph grammars—basic definitions, normal forms, and complexity. *Info. Control 69*, 136–167.

ROZENBERG, G. (Ed.) 1997. *Handbook on Graph Grammars and Computing by Graph Transformation: Foundations*. World Scientific.

SCHÜRR, A. 1994. Specification of graph translators with triple graph grammars. In *Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science*. 151–163.

SCHÜRR, A., WINTER, A. J., AND ZÜNDORF, A. 1999. The PROGRES approach: Language and environment. In *Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*. H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, Eds. World Scientific, 487–550.

SOFFER, A. AND SAMET, H. 1998. Pictorial query specification for browsing through spatially-referenced image databases. *J. Visual Languag. Comput. 9*, 6, 567–596.

WEITZMAN, L. AND WITTENBURG, K. 1993. Relational gramamrs for interactive design. In *Proceedings of the IEEE Symposium on Visual Languages*. 4–11.

WITTENBURG, K. 1992. Earley-style parsing for relational grammars. In *Proceedings of the IEEE Workshop on Visual Languages*. 192–199.

YANG, Y. D. AND ZHANG, H. J. 2001. HTML page analysis based on visual cues. In *Proceedings of the 6th International Conference on Document Analysis and Recognition*. 859–864.

YU, S. P., CAI, D., WEN, J. R., AND MA, W. Y. 2003. Improving pseudo-relevance feedback in Web information retrieval using Web page segmentation. In *Proceedings of the International Conference on World Wide Web*. 11–18.

ZHANG, D. Q. 1998. Generation of visual programming languages, Ph.D. Thesis, Macquarie University.

ZHANG, D. Q., ZHANG, K., AND CAO, J. 2001a. A context-sensitive graph grammar formalism for the specification of visual languages. *Comput. J. 44*, 3, 187–200.

ZHANG, K., ZHANG, D. Q., AND DENG, Y. 2001b. Graphical transformation of multimedia XML documents. *Ann. Soft. Engin. 12*, 1, 119–137.

ZHANG, K., ZHANG, D. Q., AND CAO, J. 2001c. Design, construction, and application of a generic visual language generation environment. *IEEE Trans. Soft. Engin. 27*, 4, 289–307.

ZHANG, K. B. AND ZHANG, K. 2002. Grammar-based approach for a visual programming language generation system. In *Proceedings of the 2th International Conference on Theory and Application of Diagrams*. Lecture Notes in Computer Science, vol. 2317, 106–109.

ZHANG, Z., HE, B., AND CHANG, K. C. C. 2004. Understanding Web query interfaces: Best-effort parsing with hidden syntax. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 107–118.

ZHANG, K., KONG, J., QIU, M. K., AND SONG, G. L. 2005. Multimedia layout adaptation through grammatical specifications. *ACM/Springer Multimedia Syst. 10*, 3, 245–260.

Received January 2004; revised September 2004, March 2005; accepted August 2005 by Brad Myers