A Framework of Syntactic Models for the Implementation of Visual Languages

G. Costagliola, A. De Lucia*, S. Orefice** and G. Tortora

Dipartimento di Informatica ed Applicazioni
Università di Salerno - 84081 Baronissi, Salerno - ITALY - {gencos, jentor}@dia.unisa.it

* Dipartimento di Ingegneria dell'Informazione ed Ingegneria Elettrica - Facoltà di Ingegneria di Benevento
Università di Salerno - 82100 Benevento - ITALY - delucia@ingbn.unisa.it

** Dipartimento di Matematica Pura e Applicata
Università di L'Aquila - 67100 L'Aquila - ITALY - orefice@univaq.it

Abstract

In this paper we present a framework of syntactic models for the definition and implementation of visual languages. We analyze a wide range of existing visual languages and, for each of them, we propose a characterization according to a syntactic model. The framework has been implemented in the Visual Language Compiler-Compiler (VLCC) system. VLCC is a practical, flexible and extensible tool for the automatic generation of visual programming environments which allows to implement visual languages once they are modeled according to a syntactic model.

1. Introduction

In the last years, much research has focused on formal models to describe the syntax of visual languages. Basically, two approaches for representing visual language sentences have been defined. One of these, the relation-based representation, represents a sentence as a set of graphical objects and a set of relations on them [43, 46]. The other, the attribute-based representation, conceives a sentence as a set of attributed graphical objects [10, 21, 34, 48]. In the first case, a visual sentence is defined by listing all the objects of the sentence and all the relations holding on them; as an example, a symbol \( r \) in vertical concatenation with a symbol \( c \) is represented as the pair \(((r, c), \text{ver_conc}(r, c))\) . In the second case, all the graphical objects must be listed together with the values of their attributes; as an example, the sentence above will be represented as \((r(0, 0), c(0, -1))\), where \((0, 0)\) and \((0, -1)\) are the coordinates of the positions of \( r \) and \( c \) in the Cartesian plane, respectively. Even though the relation \( \text{ver_conc} \) is not explicit in the input representation its presence is implicit and can always be verified on the attribute values.

Based on the attribute-based approach, the syntax of visual languages can be characterized depending on the structure of their graphical objects and on the way they can be composed in order to form visual sentences. As an example, an icon-oriented visual language is based on a set of primitive icons, each one characterized by the coordinates of the upper-left and lower-right points of its bounding box, and implicit relations (such as horizontal and vertical concatenation) to spatially arrange those icons and form iconic sentences.

In this paper we present a framework of syntactic models for visual languages. For each model we provide its syntactic features by specifying the attributes for a generic language object and the implicit relations that can act on them. It should be clear that syntactic models and visual languages are independent. We show that the same visual language may be syntactically modeled in different ways. The choice of the more appropriate model for a visual language depends on how naturally it describes that language and on the needs of the visual language implementor.

The proposed framework allows a hierarchical characterization of visual language environments naturally implemented through the object-oriented technology. This has lead to an easy integration of the syntactic models in the implementation of the Visual Language Compiler-Compiler (VLCC) system [10]. VLCC is a grammar-based graphical system that inherits, and extends to the visual field, concepts and techniques of compiler generation tools like YACC [27].

2. Visual language syntax

A visual language may be conceived as a collection of visual sentences given by graphical objects which are arranged in two or more dimensions. Visual languages are syntactically described by the graphical objects of the language (the vocabulary), the relations used to compose
the sentences, and a set of rules defining the set of visual sentences belonging to the language. The graphical objects of a visual language vocabulary are characterized by a set of attributes. Attributes of a graphical object can be distinguished in graphical attributes, syntactic attributes and semantic attributes.

Graphical attributes characterize the image of the object. Typical graphical attributes are position, size, shape, color, name, etc. Syntactic attributes are used to relate graphical objects in order to form visual sentences. A set of graphical objects form a visual sentence once all the syntactic attributes have been instantiated. Syntactic analysis techniques can be used to check whether the syntactic attributes of the graphical objects in a visual sentence are instantiated according to a proper set of relations, i.e., to check whether a visual sentence belongs to a visual language. Semantic attributes are used to associate semantics with a graphical object; they can be used either to translate a visual sentence into a target format in a syntax-direct fashion or to execute a visual sentence.

The type of syntactic attributes associated with the graphical objects of a visual language and the type of relations that can be used to compose visual sentences are strongly related and define a syntactic model. As an example, let us consider the visual sentence in Figure 2.1 representing a flowchart. It can be modeled as the interconnection of the graphical objects start, predicate, function, and halt.

Each object has a set of pre-defined attaching points as syntactic attributes. The attaching points are connected through polylines which visually depict the control-flow relation among objects. In this case, the semantics associated with the attaching points of graphical objects implicitly defines the direction of the connections.

If a, b and c indicate the three interconnections (start, predicate, function), (predicate, function) and (predicate, function), respectively, the syntactic attributes of the objects forming the sentence in Figure 2.1 are instantiated as in Table 2.1.

The two examples shown in Figures 2.1 and 2.2, respectively, reflect the two basic modalities that can be used to compose visual sentences, either connecting or spatially arranging graphical objects. In the first case, the syntactic attributes of graphical objects are instantiated by means of link relations (the connections between them). In the second case the syntactic attributes are automatically instantiated once a graphical object is placed in the Cartesian plane, while the relations between two objects are implicitly derived from their relative positions.

It is worth noting that the same visual language can be syntactically modeled in different ways. For example, depending on the type of syntactic attributes and relations considered, the graph shown in Figure 2.3 can be modeled either by explicitly interconnecting nodes and arrows (graphical objects) between them or by spatially arranging them in the space, in such a way the position of an end point of an arrow coincides with the coordinates of a point on the circumference of a node.

<table>
<thead>
<tr>
<th>object name</th>
<th>attaching point 1</th>
<th>attaching point 2</th>
<th>attaching point 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>a</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>predicate</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>function</td>
<td>b</td>
<td>a</td>
<td>-</td>
</tr>
<tr>
<td>halt</td>
<td>c</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.1. An alternative representation of the flowchart in Figure 2.1.

In the second case, the implicit relations among nodes and arrows are geometrically derived. The choice of the
syntactic model used to specify a visual language type depends on its intrinsic characteristics: for example, for graph languages the connection based syntactic model is more natural than the geometric based model.

In addition to the syntactic attributes and relations provided by the particular model, in general, each visual language is associated with a special relation named `annotate` which allows graphical objects to be associated to a visual sentence. As an example, in the flowchart in Figure 2.1 the relation `annotate` allows to associate each node with a piece of code describing the action or condition to be executed. This relation is different from the inclusion relation since it establishes a relationship which is logical but not spatial and, if used recursively, it allows to create nested visual languages.

### 3. Syntactic models for visual languages

In this section we give a hierarchy of syntactic models. This hierarchy is not meant to be exhaustive but it is able to provide a wide characterization of existing visual languages. Table 3.1 summarizes the descriptions of the basic syntactic models which are described in this section.

<table>
<thead>
<tr>
<th><strong>Visual syntactic model</strong></th>
<th><strong>relations</strong></th>
<th><strong>syntactic attributes</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection-based</td>
<td>any connection</td>
<td>a pre-defined number of sets of attaching points</td>
</tr>
<tr>
<td>Geometric-based</td>
<td>any spatial composition</td>
<td>a set of points of the graphical object</td>
</tr>
<tr>
<td>Plex</td>
<td>any connection</td>
<td>a pre-defined number of attaching points</td>
</tr>
<tr>
<td>Box</td>
<td>any spatial composition</td>
<td>the upper-left and the lower-right points of the box</td>
</tr>
<tr>
<td>Iconic</td>
<td>any spatial concatenation plus overlapping</td>
<td>the upper-left and the lower-right points of the icon</td>
</tr>
<tr>
<td>Symbolic</td>
<td>spatial concatenation</td>
<td>the coordinates of the symbol</td>
</tr>
<tr>
<td>String</td>
<td>string concatenation</td>
<td>the position of the symbol in the string</td>
</tr>
</tbody>
</table>

Table 3.1. Basic classification of syntactic models for the visual languages.

Figure 3.1 shows how the syntactic models are hierarchically related. Note the hybrid syntactic model `box-and-graph` derived by the basic syntactic models Connection-based and Box.

![Figure 3.1. A hierarchy of syntactic models](image)

### 3.1 The connection-based models

**Connection-based**

The visual sentences described by this syntactic model are formed by a set of interconnected objects. The syntactic attributes of a object can be single attaching points or sets of attaching points (attaching areas). The value of an attaching point participating in a connection is given by a unique identifier for that connection. There may be many types of connections among the attaching points of objects and in this paper we report three of them (see Figure 3.2): attaching points can be connected point-to-point (a), or connected by plugging protrusions ("knobs") into correspondingly shaped indentations.
three types of connection may be used, simultaneously.

The selection of a connection type is done depending on the characteristics of the visual language under study. As an example, edge-labeled graphs and directed graphs may be modeled as sets of nodes and edges where the circumference of a node (attaching area) is connected "point-to-point" to at least one end (attaching point) of an edge. The choice of this representation takes into account the fact that, in these cases, an edge does not only connects nodes but also conveys information about labels and direction. On the other hand, undirected graphs can be modeled either as sets of nodes and edges (graphical objects) interconnected "point-to-point", as in the previous case, or as sets of nodes joined through polylines ("links") representing relationships between nodes.

Note that for a visual language, if needed, all the three types of connection may be used, simultaneously.

Figure 3.2. Three types of connections: point-to-point (a), puzzle (b) and link (c).

The connection-based model may be used to describe the syntax of a number of graph languages. Examples include languages based on data flow graphs, state transition diagrams, Petri nets and entity-relationship diagrams.

In data flow graph languages the operations are typically associated to boxes and the data flows along the arrows connecting them. In general, graphical objects of such languages may present a variable number of attaching points (attaching area), as in the case of Data Flow Diagrams used for specifying the data flow in software systems, or in the case of PegaSys [35] which is a diagrammatic system designed to provide a programming environment where all steps of software life cycle are supported by graphical interaction languages. Phred [5] is a visual parallel programming environment that allows a software designer to create Phred programs, to statically analyze them for determinancy, and to interpret them. A Phred program is composed of a control flow graph, a data flowgraph, and a set of node interpretations represented by procedure specification (written in C language).

Transition diagrams of finite state automata are another typical example of connection-based languages. The graphical objects of such languages are nodes and arcs representing the states and the transitions of the automaton, respectively. The attaching area of a node is given by its circumference, while an arc has two attaching points corresponding to its end-points. An example of language based on state transition diagrams has been proposed by Jacob [26] for specifying user interfaces in a graphical manner.

Many visual programming systems are based on the Petri nets formalism for specifying concurrent and real-time systems [1]. For example, the MOPS-2 system [2] uses coloured Petri nets to allow parallel systems to be constructed and stimulated in a visual manner. The VERDI system [22] uses a form of Petri nets for specifying and simulating distributed systems: the specification is animated by moving tokens around the network. Cabernet [40] is a visual environment based on high-level Petri nets [18] designed to support the specification and verification of real-time systems.

Plex

The syntactic model Plex is a sub-type of the connection-based model. Here, attaching areas are not allowed. The languages corresponding to this category were first defined by Feder in [16].

The plex model can be used to describe the syntax of visual languages like flowcharts, chemical structures, Boolean and electric circuits, and so on. For instance, in flowchart languages each graphical object (boxes, diamonds...) has a pre-defined number of attaching points (two for boxes, three for diamonds,...), and graphical objects can be connected only through links visualized as polylines. Examples of flowchart-based languages include FPL (First Programming Language) [13], particularly suited to help novices in learning programming since it eliminates syntactic errors, OPAL [36] which allows doctors to enter knowledge about cancer treatments into an expert system, GAL [3] and PIGS [42] that use a flowchart variant called Nassi-Shneiderman flowcharts [37], and Pict [19] which uses conventional flowcharts, but is differentiated by its use of colour pictures rather than text inside the flowchart boxes.

Also data flow graph languages whose boxes have a fixed number of attaching points can be modeled according to the plex syntax. This is the case, for example, of PROGRAPH [41] which is a structured, functional language introduced to solve comprehension problems of conventional textual functional languages, and InterCONS [45] which is a data flow system developed in Smalltalk that supports programming by example.

Nested data flow graph languages can be also modeled according to the plex syntax, where the nesting is obtained by using the relation annotate. Usually, in a nested data-flow graph each node represents an operator (function), an operand (variable), or another directed graph represented by an iconic name. Examples include the programming by example system Fabrik [32] developed in Smalltalk and used for constructing user interfaces and the commercial product Lab-VIEW [38] running on Macintosh and used for controlling external devices. Lab-
VIEW provides procedural abstraction, control structures, and many useful primitive components such as knobs, switches, and mathematical functions. Show and Tell [29, 39] is a visual programming language for defining interfaces of relational databases (schema and queries). A Show and Tell database schema is defined by a two-dimensional layout of base boxes, while a query is defined by a data flow program.

Other general purpose nested data flow visual programming language systems are Hyperflow [30] and its derivatives ProtoHyperflow [17] and HF/PP [31]. The syntax of these languages consists of boxes and arrows, a box representing a process, and an arrow representing a data flow between processes. Boxes may contain data flow graphs for defining new functions and to build conditionals. Hyperflow and HF/PP are designed as visual languages for a pen-based multimedia system, while ProtoHyperflow is implemented on a traditional mouse/CRT-based system.

Another visual language that can be modeled by plex syntax has been recently proposed by Karsai [28]. It consists of a configurable visual programming environment that can be customized for various application domains.

The BLOX languages proposed by Glinert [20] are another example of plex languages. Visual sentences are composed by joining graphical objects using the usual jigsaw-puzzle "lock and key" metaphor to plug protrusions ("knobs") into correspondingly shaped indentations ("sockets") so that the two juxtaposed tiles interlock. Moreover, BLOX substructures can be encapsulated (nested) into BLOX elements. Languages for representing logical circuits and structured programs (in a similar way to nested flowchart, where function boxes may hide sub-flowcharts) have been defined using the BLOX methodology.

3.2 The geometric-based models

Geometric-based

In this model, a visual sentence is described as a set of graphical objects spatially arranged in the Cartesian plane. The syntactic attributes of a graphical object are the coordinates of a set of representative points on its image. Syntactic attributes are automatically instantiated when placing the graphical object in the Cartesian plane. Several spatial composition rules can be used to form visual sentences. Examples include metrics based relations, such as horizontal and vertical concatenation, spatial inclusion, adjacency and intersection. As an example Venn diagrams used to depict set relations, such as inclusion and intersection, can be modeled according to the geometric-based syntactic model (see Figure 3.3).

The geometric-based model can be used to describe high level interface languages for geographical information systems. For example, Cigales [4] is a graphical query language based upon spatial query by drawing a pattern according to the user's mental model of the data to be retrieved. The graphical objects of this language are geometric objects, such as lines and areas, while queries (sentences) are composed by means of spatial rules, such as inclusion, intersection, and adjacency.

Box

This is a special case of the geometric-based syntactic model. Here an object must be described by a square or a rectangle. The set of representative points used as syntactic attributes is given by the upper-left and lower-right points of the box. Visual sentences are composed using the same spatial relations as in the geometric-based model. As an example, the syntactic model Box can be used for visual languages describing document layouts (see Figure 2.2), descriptions of structured programs, etc.

Iconic

This model is another specialization of the geometric-based one. The main difference is that the concept of area is not defined on icons, thus preventing the use of spatial relations, such as intersection and inclusion, in the composition of iconic sentences. The spatial relations used to form iconic sentences are overlapping and geometric based relations. The syntactic attributes of an icon are the coordinates of the upper-left and lower-right points of its bounding box.

Figure 3.4 shows two simple iconic sentences: the first one, taken from the Heidelberg Icon Set [44], describes the operation "delete string", while the second one, taken from [12], represents the operation "display a text file".
users to create multi-layered iconic annotations of streams video data. The SIL-ICON [6] compiler is a system for specifying, interpreting and prototyping icon-oriented systems. It uses context-free grammar augmented with spatial operators to define a visual language and allow iconic sentences to be constructed by arranging icons in a two-dimensional fashion and parsed and interpreted according to the specification rules of the language. Minspeak [7] is an iconic language system used in augmentative communication by people with speech disabilities. It consists of multi-meaning one-dimension iconic sentences which are used to retrieve messages (words or word sequences) stored in the memory of a microcomputer. A built-in speech synthesizer allows to generate the voice output.

**Symbolic**

The symbolic model is a specialization of the iconic one. The only syntactic attributes of a symbol are the coordinates of its position in the space, i.e., the coordinates of a representative point, such as its centroid. The spatial relations that can be used to compose symbolic sentences are the same as in the case of iconic sentences, with the exception of overlapping. As an example, some symbolic languages for two-dimensional arithmetical expressions can be found in [9, 10].

**String**

This model directly derives from the symbolic one. Here a graphical object is a character whose only syntactic attribute is its position in a one-dimensional space (string). The only relation is string concatenation. This syntactic model is suitable for all the string languages.

### 3.3 The hybrid models

**Box-and-graph**

Box-and-graph is an example of hybrid syntactic model obtained by grouping features of the box and connection-based models. A graphical object has both the upper-left and lower-right points of its bounding box and a pre-defined number of attaching areas and/or attaching points as syntactic attributes. The relations that can be composed to form visual sentences are both geometric (spatial inclusion, horizontal and vertical concatenation, etc.) and connection based. Statecharts [23, 24] are an example of language that can be modeled as box-and-graph. Statecharts are an extension of state transition diagrams introduced to specify large and complex reactive systems, that is, event-driven continuously reacting to stimuli systems.

The main features of statechart transition diagrams are AND/OR decomposition of states, inter-level transitions, and a broadcast mechanism for communication between concurrent components. Graphical objects representing states or AND/OR decomposition of states may contain nested statecharts. Examples of visual languages based on statechart formalism include Miro [33] which is a language for defining security constraints (in particular user accesses to files) in operating systems, and StateMaster [47] which is a visual system for programming graphical user interfaces.

### 4. Conclusions

We have presented a framework of syntactic models for visual languages and provided a wide characterization of existing visual languages by modeling each language into what we feel be the more appropriate syntactic model.

The proposed framework underlies a hierarchical characterization of visual language environments which has guided the object-oriented implementation of the Visual Language Compiler-Compiler (VLCC) system, [10]. VLCC is a tool for the automatic generation of visual programming environments implementing visual languages. A prototype of the VLCC system has been implemented using object-oriented technology under MSWindows™3.x and MSWindows95/NT™. VLCC is able to implement a visual language according to its syntactic model. Thanks to the modular structure of the system together with its object-oriented architecture, it is possible to easily implement new syntactic models.

Figures 4.1 (a), (b) and (c) show three VLCC generated visual programming environments implementing, respectively, the visual languages boxes (box), semi-structured flow charts (plex) and entity-relationship (connection-based). The last two languages are actually hybrid languages since they allow a user to annotate an object on the screen with string sentences. This is done by selecting the TEXT button and clicking on the chosen graphical token. This action launches a text editor for the insertion of a string sentence (see Figure 4.1 (b)). In the figures, the results of the compile processes are also shown. In (a), a set of interrelated boxes are recognized as a sentence of a simple box language; in (b) a semi-structured flowchart is translated into Pascal-like code; finally, in (c) and (d), an entity relationship diagram and its translation into SQL code are shown.

More details on VLCC tool and the visual languages generated can be found at the WWW site: [http://www.unisa.it/gencos.dir/vlcc.htm](http://www.unisa.it/gencos.dir/vlcc.htm)

As further research, we are focusing on the definition and implementation in VLCC of syntactic models for the description of dynamic visual languages, [8], where graphical object attributes for handling temporal information need to be introduced.
References
