Visual Parsers based on Extended Constraint Multiset Grammars

Jiro Tanaka
Institute of Information Sciences and Electronics
The University of Tsukuba
Tsukuba, Ibaraki 305-8573, Japan
jiro@is.tsukuba.ac.jp

Abstract— **We are working for visual paring in these years. We have developed the series of visual parser generators, such as Eviss, VIC and Rainbow. They generate a spatial parser by defining the grammars of visual language. Using generated spatial parser, they can analyze the figures and can execute the specified actions. The GUI generator and the subset of VISPATCH are shown as examples.**

Keywords— **Graphical user interface, Visual parser, Visual system, Constraint solving**

I. Introduction

THOUGH the textual paring has already been well-established in computer science field, the visual paring is still in the preliminary stage.

Visual languages are used in various fields, such as ER diagrams, object diagrams of OMT, formula, music, and diagrams that show relationships between characters appearing in TV dramas. Visual languages have structures like textual languages.

We can assume the diagrams which represent graph structures as a visual language. A special purpose graphic editor can be considered as a system to process a visual language. We call a system which processes a visual language a *visual system*. Visual systems proposed so far have been fixed on certain specifications. It was a difficult and time consuming job to modify those systems.

Therefore, we are working for general purpose visual systems in these years. We have developed the series of visual parser generators, such as Eviss [1], [2], VIC [3] and Rainbow [4], [5].

II. The extended Constraint Multiset Grammars

We use the extended Constraint Multiset Grammars (CMG) [6], [7] in defining the grammars of visual system in visual parser generators. CMG consist of a set of terminal symbols, a set of non-terminal symbols, a distinctive start symbol, and a set of production rules. The terminal and non-terminal symbols have various attributes. The production rules are used to rewrite a multiset of tokens (the instances of the terminal or non-terminal symbols) for a new symbol.

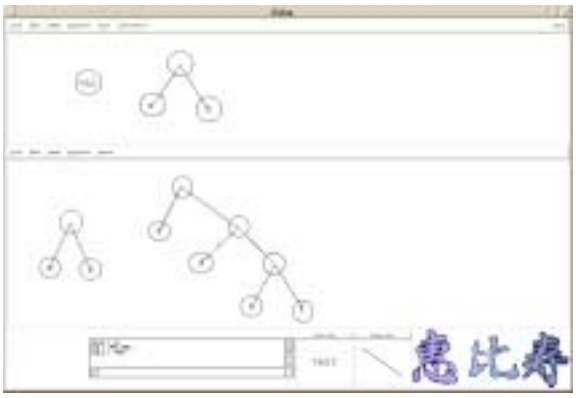

Fig. 1. A snapshot of Eviss

The constraints maintain the relationships between the attributes of the tokens. A production rule is defined as follows:

$$\begin{aligned} T(\vec{x}) \quad ::= \quad & T_1(\vec{x_1}), \cdots, T_n(\vec{x_n}) \ where \\ & exists\ T'_1(\vec{x'_1}), \cdots, T'_m(\vec{x'_m}) \\ & where\ C\ and\ \vec{x}\ =\ F\ and\ A \end{aligned}$$

When the attributes of the tokens $T_1, \cdots, T_n$ ("normal" components) and $T'_1, \cdots, T'_m$ ("exist" components) satisfy the constraints C, the tokens $T_1, \cdots, T_n$ are rewritten to the non-terminal symbol T. Exist components are needed to recognize T and are not rewritten to T [2]. F is the function that has the attributes $\vec{x_1}, \cdots, \vec{x_n}$ and $\vec{x'_1}, \cdots, \vec{x'_m}$ of the components as arguments, and the return value of the function is given to the non-terminal symbol T as its attribute.

Note that we have extended the original CMG to include action A. A is defined as "script program executed when the production rule is applied." In the extended CMG, we can specify arbitrary actions, such as computing values and rewriting figures.

[1]This paper has been extracted from: Jiro Tanaka, "Visual Parsing and 3D Visual Interface," *Proceedings of 2000 International Conference on Information Society in the 21st Century (IS2000)*, 2000.

[2]CMG also has "not_exist" and "all" components. For details, refer to [2], [6], [7].

III. List Tree example

List Tree is defined recursively by the following two production rules.

Rule 1: A non-terminal symbol "list" consists of a "circle" and a "text" in the center of it.

Rule 2: A non-terminal symbol "list" consists of a "circle," two "lines" and two "lists." The two "lists" are connected to the "circle" by the "lines."

These production rules can be written by the extended CMG as follows.

```
1:  list(point mid, integer mid_x,
2:                        string value) ::=
3:     C:circle, T:text
4:     where (
5:        C.mid == T.mid
7:     ) and {
8:        mid = C.mid
9:        mid_x = C.mid_x
10:       value = {script.string {
11:                       list @T.text@}}
12:    } and {
13:       display(value = @value@)
14:    }
15:
16: list(point mid, integer mid_x,
17:                       string value) ::=
18:    C:circle
19:    exists S1:list, S2:list,
20:                       L1:line, L2:line
21:    where (
22:       S1.mid == L1.end
23:       S2.mid == L2.end
24:       C.mid == L1.start
25:       C.mid == L2.start
26:       C.mid == T.mid
27:       S1.mid_x < S2.mid_x
28:    ) and {
29:       mid = C.mid
30:       mid_x = C.mid_x
31:       value = {script.string {
32:                  concat [list @S1.value@]
33:                         [list @S2.value@]}}
34:       lef = C.lu_x
35:       right = C.rl_x
36:    } and {
37:       display(value = @value@)
38:    }
```

Lines 1 to 14 show the definition of the production rule 1. Line 1 shows the attributes of the non-terminal symbol "list." Attributes are "mid," "mid_x,"and "value." Line 3 shows that this non-terminal consists of a "circle" and a "text" string and these components are "Normal." At line 5, constraints are defined. This line shows that the attribute "mid" of "circle" C is equal to the attribute "mid" of "text" T. "mid" is an attribute that indicates the center's coordinates. In lines 8 to 11, the values of Attributes are defined. Line 8 shows that an attribute "mid" of "list" is equal to "mid" of "circle" C. Line 9 shows that an attribute "mid_x" of "list" is equal to "mid_x" of "circle" C. "mid_x" is an attribute indicating an abscissa. Lines 10 and 11 show the definition of an attribute "value."

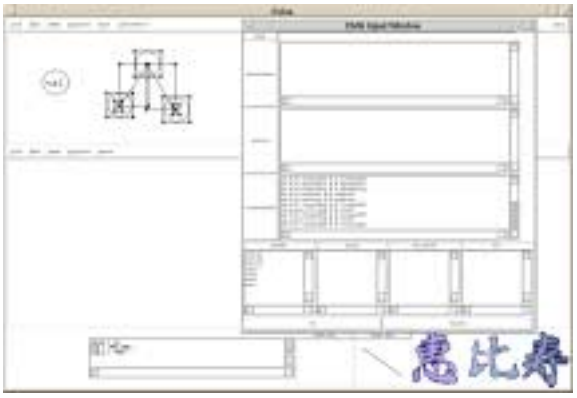

Fig. 2. Defining grammars in Eviss

This definition represents that "text" string of "text" T is treated as a list. At the line 13, action is defined. This line shows output (*value*) when this production rule is applied.

Lines 16 to 38 show the definition for the production rule 2. Lines 19 and 20 show that two "lists" and two "lines" must exist somewhere in the visual sentence. Line 27 shows that "list" S1 is on the left side of "list" S2. This constraint distinguishes the left "list" from the right "list." Lines 31 to 35 show the definition of the attribute "value." Here, two "lists" S1 and S2 are connected.

IV. Eviss

We made the visual system Eviss [1], [2] which has a spatial parser generator. Figure 1 shows the snapshot of a visual system that represents "List Tree." The upper half of the screen is called the *definition window*. A person who implements a visual system defines grammars of visual languages in the definition window. The bottom half is called the *execution window*.

In Eviss, figures are used to define rough grammars. At first, the user draws figures which he wants to define as a new non-terminal symbol from the definition window. We call these figures as "example figures." Eviss automatically extracts simple constraints and components from "example figure" and outputs to the CMG Input Window with text. Then the user edits the constraints and components in the CMG Input Window (Figure 2). The user can also specify actions in this phase.

At the *execution phase*, a user draws figure elements which should be analyzed to the execution window.

V. Examples of making visual systems in Eviss

A. The GUI Creator

We define expressions for widgets and binding in the visual language. A frame widget (`Frame`) is repre-

Fig. 3. An example of GUI.

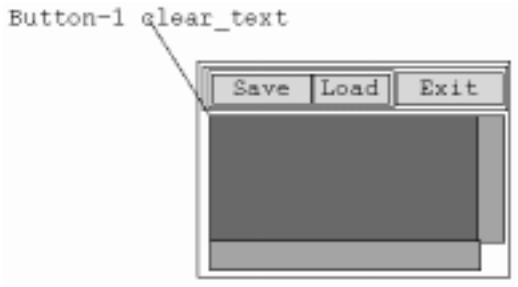

Fig. 4. Visual program that represents Figure 3.

sented by a rectangle which is not painted. A scroll bar widget (`Scroll`) is represented by a rectangle painted with orange. A text widget (`TextW`) is represented by a rectangle painted with red. A button widget (`Button`) is represented by a rectangle painted with yellow and a text string in it.

We show an example of describing a GUI in the visual language. Figure 3 is an example of a GUI and Figure 4 is the visual program which represents it. Suppose that the procedure `clear_text` is called when left button of the mouse is clicked on the text widget in Figure 3. The binding appears in the visual program though it does not actually appear on the screen. This is because GUIs do not consist of only visual informations.

We have defined production rules for creating a GUI by combining widgets. The production rule for `Binding` has a text string and a line as its components. In figure 4, the text string is "`Button-1 clear_text`" and the `GUI` is the text widget. The text string is a list which consists of the name of the event and the name of the procedure which is called when the event occurs. In Figure 4, the event is `Button-1` and the name of the procedure is `clear_text`.

B. The subset of VISPATCH

VISPATCH [8] is a visual system which redraws figures according to rules represented by figures. Redrawing is started by events caused by users or the system such as mouse clicks and drags.

We have implemented the subset of VISPATCH in

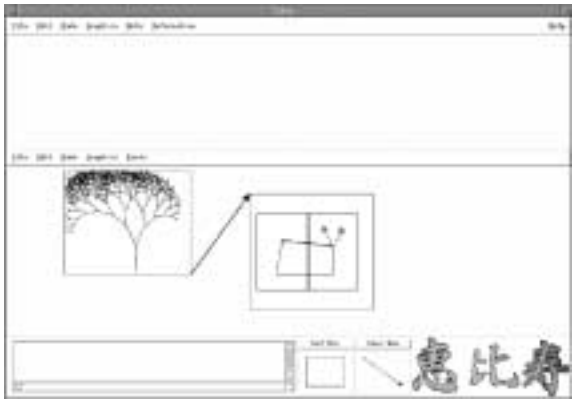

Fig. 5. Snapshot of VISPATCH implemented in Eviss.

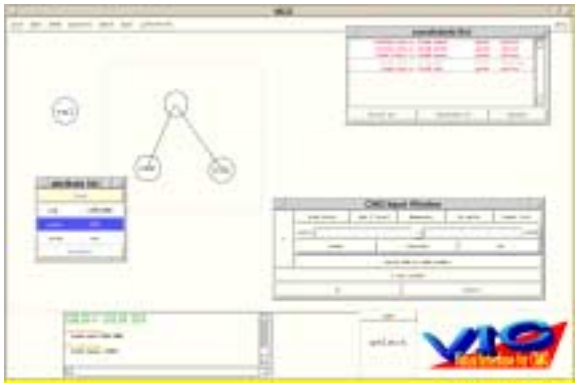

Fig. 6. A snapshot of VIC

Eviss (Figure 5). VISPATCH starts redrawing by events, in other words, VISPATCH is *event driven*. Eviss starts spatial parsing (and redrawing) by drawing, deleting and altering figure elements, in other words, Eviss is *data driven*. To start spatial parsing, if an event occurs in the event sensor, a figure element that is the same as in the rule head is drawn in the event sensor. After spatial parsing is finished, a procedure which creates production rules for VISPATCH rules is called in action.

VI. VIC

VIC [3] is the successor of Eviss. There are two main differences between Eviss and VIC. The first difference is that Eviss has two windows, i.e., Definition Window and Execution Window, and only Execution Window has a spatial parser. Whereas, VIC has only one window. Because of having one window only, VIC has no border between Definition Window and Execution Window. VIC can understand the non-terminal symbols when the user defines the grammars.

The second difference is that VIC can define constraints by the direct manipulation of "example figures" without using CMG Input Window. In Eviss, user had to input CMG textually from CMG Input

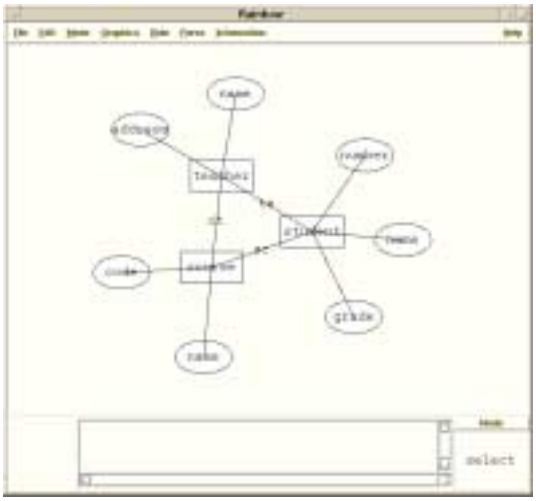

Fig. 7. Example of the E-R diagrams

Window. This made difficult to define the visual system intuitively. In the case of VIC, the user can define various visual systems intuitively, even if the user does not know the grammar of CMG.

The snapshot of VIC is shown in Figure 6.

VII. Rainbow

The visual system must have layout capability, since it performs actions such as creating, deleting, and moving the figures. Even if a user lays out a portion of the figure, the entire figure can be hard to understand. It is important to make the entire figure more balanced and understandable.

We therefore developed Rainbow[4], [5], a visual system generator that can handle layout constraints. The system can interactively layout whole figures while parsing them, and make the parsed figures more balanced and understandable. Rainbow was implemented by adding the layout constraints to Eviss.

Using Rainbow makes it possible to more interactively handle figures, such as the various diagrams that are used in the software engineering field. Figures can be interactively laid out while they are parsed by adding layout capability to their spatial parser, and the parsed figures are more understandable. Rainbow is therefore a useful tool for making CASE tools.

The snapshot of Rainbow in the E-R diagrams example is shown in Figure 7.

VIII. Acknowledgments

The author would like to express thanks to Akihiko Baba, SackTae Joung, Kenichirou Fujiyama and Kazuhisa Iizuka for their cooperation in the project. The work reported in this paper has been based on their research efforts.

References

[1] Akihiro Baba and Jiro Tanaka: "A Visual System Having a Spatial Parser Generator," *Transactions of IPSJ*, Vol. 39, No. 5, pp. 1385–1394, 1998, *in Japanese*.

[2] Akihiro Baba and Jiro Tanaka: "Eviss: a Visual System Having a Spatial Parser Generator," *Proceedings of Asia Pacific Computer Human Interaction*, pp. 158–164, 1998.

[3] Kenichirou Fujiyama, Kazuhisa Iizuka and Jiro Tanaka, "VIC: CMG Input System Using Example Figures," *Proceedings of the International Symposium on Future Software Technology*, pp. 67–72, 1999.

[4] SackTae Joung and Jiro Tanaka: Rainbow: "Implementing Layout Constraints in Visual System Generator," *Transactions of IPSJ*, Vol.41, No.5, pp. 1246-1256, 2000, *in Japanese*.

[5] SackTae Joung and Jiro Tanaka: "Generating a Visual System with Soft Layout Constraints," *Proceedings of the International Conference on Information – Information'2000 –*, Fukuoka, Japan, Oct. 16-19, pp. 138-145, 2000.

[6] Kim Marriott: "Constraint Multiset Grammars," *Proceedings of the IEEE Symposium on Visual Languages*, pp. 118–125, 1994.

[7] Sitt Sen Chok and Kim Marriott: "Automatic Construction of User Interfaces from Constraint Multiset Grammars," *Proceedings of the IEEE Workshop on Visual Languages*, pp. 242–249, 1995.

[8] Yasunori Harada, Kenji Miyamoto and Rikio Onai: "VISPATCH: Graphical rule-based language controlled by user event," *Proceedings of the 1997 IEEE Symposium on Visual Languages*, pp. 162–163, 1997.