A Data-Flow Visual Approach to Symbolic Computing: Implementing a Production-Rule-Based Programming System through a General-Purpose Data-Flow VL

Mauro Mosconi - Marco Porta
Dipartimento di Informatica e Sistemistica – Università di Pavia
Via Ferrata, 1 – 27100 – Pavia – Italy
mauro@vision.unipv.it - porta@vision.unipv.it

Abstract

The main aim of this paper is to investigate how the production-rule-based computational paradigm can be implemented through visual data-flow techniques. We propose a simple yet effective system for visually composing rule preconditions and actions, through a general purpose data-flow visual language.

1. Introduction

Among general purpose visual programming systems, data-flow ones are very widespread, certainly also thanks to their simple and intuitive functioning mechanism. It is very interesting to note that the data-flow model may be viewed as a generalization of the event-driven programming model [1]: each node waits for data (events) to arrive and then fires. Hence, production-rule systems can also be viewed as data-flow systems, in which rule conditions act as demons awaiting the arrival of certain data elements before executing their conclusion.

Although production systems are the basis for many expert systems, their functioning mechanism may be especially difficult to understand for those who are used to programming according to the control-flow (imperative) paradigm. Building correct productions to solve a particular problem is a task which may require many attempts by an unskilled programmer (addition and elimination of conditions and/or actions) and a textual-only rule representation may turn out to be very unintuitive. Very often, in fact, different rules hold the same preconditions, and comprehension of the way they affect the global system may depend on just such a sharing.

This paper will show how it is possible to easily implement the production-rule-based computational paradigm by using visual data-flow techniques. The simple but effective system we will describe, which is based on VIPERS [2], a general-purpose data-flow visual language, is primarily intended as a learning tool and can greatly simplify the task of the novice production-rule programmer. Moreover, *we believe that our approach is particularly interesting from a theoretical point of view.*

2. System Description

We propose two substantially equivalent ways to visually build productions (see [3] for a detailed description). The former is shown in Figure 1 and better highlights distribution of conditions being shared by different rules. The latter, whose scheme cannot be shown here due to lack of space, gives up this opportunity but is probably more readable.

2.1. A first method for building productions

Figure 1 shows an example scheme composed of three productions. C*ondition* blocks C and T*est* blocks T represent rule preconditions and, when activated, give out symbolic lists denoting them. To achieve correct synchronization, VIPERS exploits control signals (thin arcs without arrows) connecting blocks control ports[1]. There is a C*ondition* C*ollector* block CC for every rule. CC blocks analyze the respective input conditions to determine whether the rule is applicable or not (that is, whether at least one production exemplar exists relative to the rule). Every CC block has another list as input, which we will call L. It is the data structure which is propagated to CC blocks outputs, after being "filled" with information deriving from precondition analysis. During the next computational cycles, it will return to the same blocks, with some changes (unless the program is finished). List L contains both past and present information and may be viewed as in-motion knowledge. The task of block CSS (C*onflict* S*et* S*olver*) is to choose which production exemplar, among the possible ones, is to be activated, according to a certain selection strategy. Several CSS blocks will be put at the programmer's disposal, so that he/she can easily select the desired strategy. Block CSS receives the various L lists emitted by CC blocks as inputs. On the basis of data contained in valid lists, CSS applies the selection strategy it represents and activates only that output signal relative to the chosen production

[1] If a signal exists between an output control port (on the right) of block A and the input control port (on the left) of block B, then block B cannot be executed before the execution of block A.

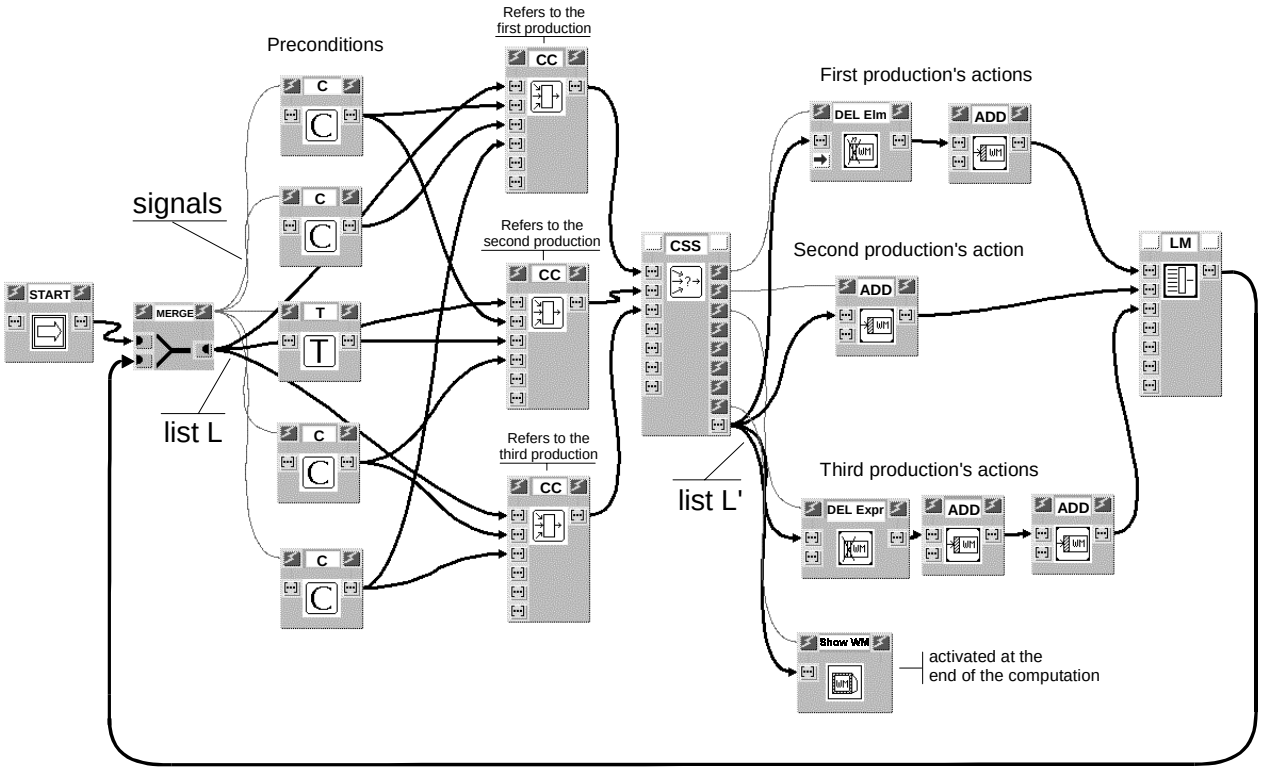

Figure 1: first scheme for building productions

action sequence or, in case of null conflict set, the last one, which powers up one or more end blocks (in Figure 1, block ShowWM, showing the working memory's contents). Moreover, CSS gives a list L' out which has almost the same structure as L and provides actions with the data to act on. The programmer can use standard action blocks (ADD and DEL, to add/delete a symbolic structure to/from working memory), plus an arbitrary number of blocks relative to actions definable according to the particular application. Every action block of the selected production receives L' as input list, properly modifies it and then gives it out. The output of the last action block executed enters block LM (L*ists* M*erge*), comes out of it without any change, goes into the initial block MERGE and, at last, submits itself to the CC blocks again. CC blocks and block CSS can be built with as many input ports as one likes.

2.2. A second method for building productions

As mentioned earlier, another way to visually specify production preconditions is possible. In this case, common conditions are repeated for each production and CS blocks are no longer used, since list L is analyzed and modified little by little by condition and test blocks themselves. In a certain sense, we may say that what was previously done by CC blocks is now accomplished in a distributed manner. Although common conditions have to be repeated, in this scheme it is probably simpler to identify and possibly add or delete conditions pertaining to a particular production.

3. Conclusion

It is important to emphasize that the described system should not be thought of as something to be utilized by a skilled programmer: of course, an ad-hoc visual environment would be much more effective. Instead, besides *being the occasion for a theoretical investigation*, it aims at being a mechanism which allows the unskilled beginner to easily try out principles and features of production-rule-based programming**.**

References

[1] Menzies, T., "Frameworks for Accessing Visual Languages", *Technical Report TR95-35*, 1996, Dept. of Software Development, Monash University, Melbourne, Australia.

[2] Ghittori, E., Mosconi, M., Porta, M., "Designing New Programming Constructs in a Data-Flow VL", in *Proceedings of the 14th IEEE Conference on Visual Languages (VL'98)*, 1-4 September 1998, Nova Scotia, Canada.

[3] Mosconi, M., Porta, M., "Implementing a Production-Rule-Based Programming System through a General-Purpose Data-Flow VL". *Technical Report*, University of Pavia, Pavia, Italy, 2000, URL: http://vision.unipv.it/research/papers/00tr-prbdfvl.html.