Specifying Control transformations through Petri Nets

Eduardo Miranda, EBAI (Argentine-Brazilian School of Informatics)

1. Introduction

In recent works on structured development of real-time systems [1,2], the need to capture control and timing information has been widely acknowledge. This has led to the introduction of control transformations and control flows into data flow models.

The tools sugested to specify control transformations are state-transition diagrams in case of sequential logic and activation or condition tables for combinatorial logic. These tools, though useful, are not enough when control transformations deals with several tasks evolving simultaneously either from the system or the user point of view, as we will see in the next examples.

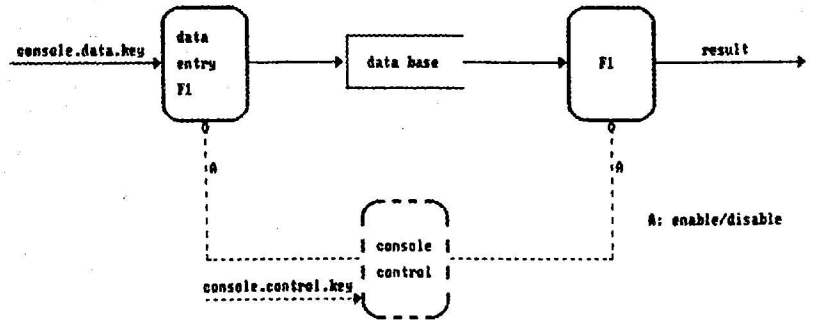

fig. 1 A simple system

Figure 1, shows a simple system composed by two data transformations and one control transformation. The data entry function stores the data entered thru the console shown in figure 2. Function "F1" processes this data and displays the results on the console display. The console control transformation,implements the console logic responsible for the activation/deactivation of the other functions.

Having just one function to represent, the console control transformation can be conveniently specified by means of the state-transition diagram of figure 3; but if we introduce a second function paralelism arises from the user point of view, as he can load

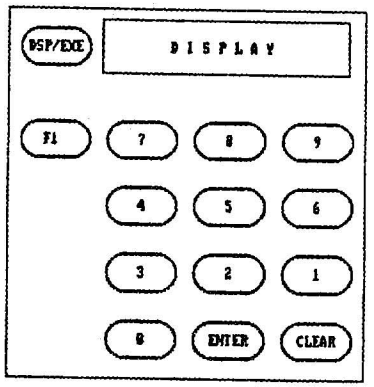

fig 2. The system console

and execute the functions in the order he wishes, becoming the state-transition diagram that of figure 4.

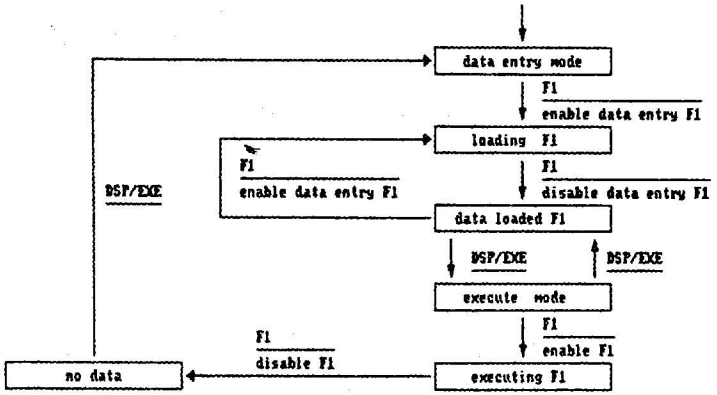

fig 3. State-transition specification for a console with one function

Generally speaking, combining n state-transition diagrams with k states each, results in a diagram with O(k**n) states. This ombinatorial explosion could be avoided using auxiliary variables to comunicate independent subsystems as in figure 5, but this reduces the readability and the graphic nature of the model.

2. Specifying through Petri Nets

Figure 6, shows the same specification by means of Petri nets. Notice that, in this case, when a new function is added to the console

the controling logic is augmented only by a replica of the subnet attending "F1". Thus, complexity of representation grows lineary being the number of places and transitions O(n).

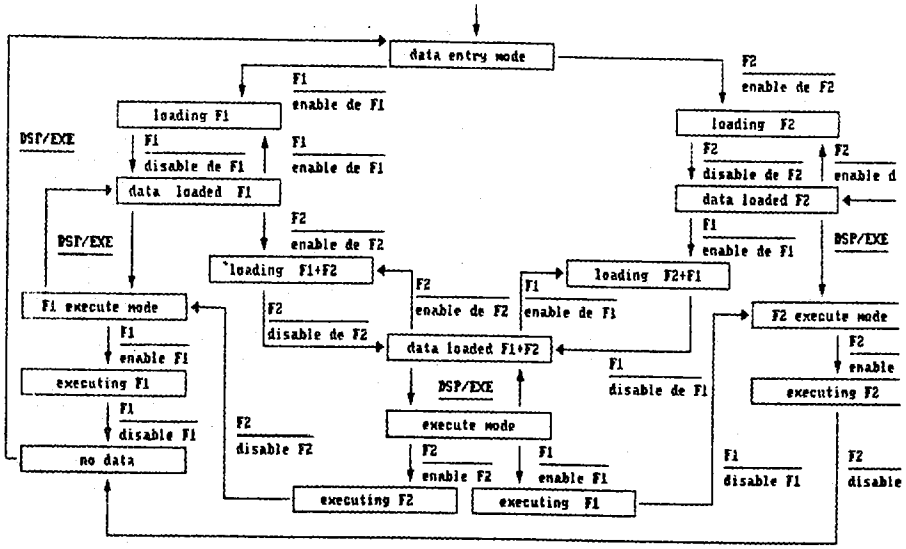

fig. 4 State-transition specification for a console with two functions

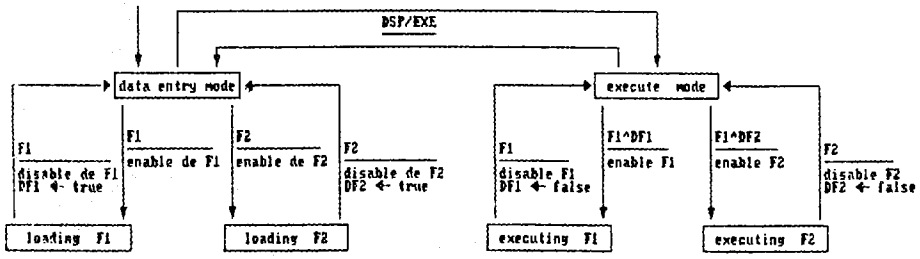

fig. 5 Using auxiliary variables in state-transition diagrams

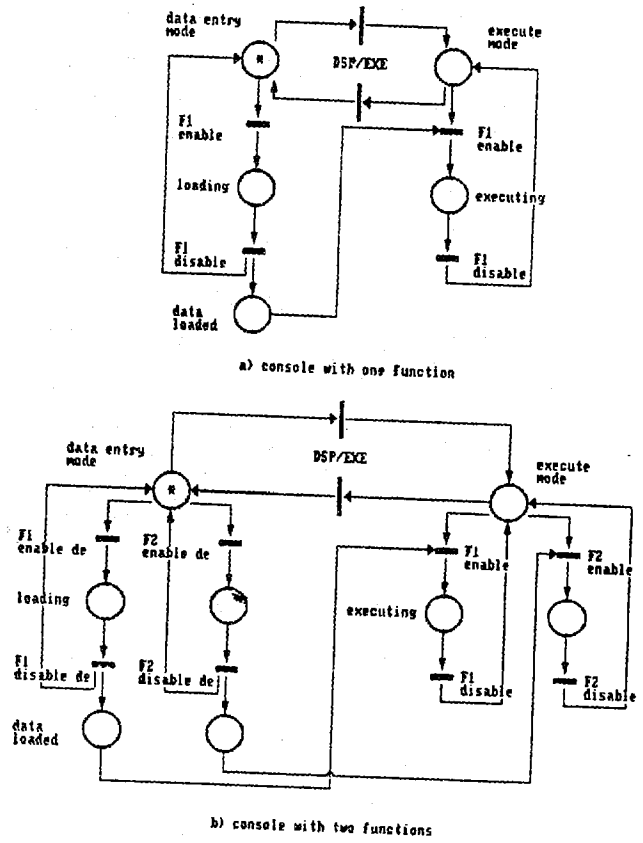

fig 6. Petri net specification for console logic

3. Abbreviations and Extensions to Petri Nets

The use of abbreviations and extensions of Petri nets, further simplifies the specification of control transformatons, satisfying the criteria that an effective and verifiable communication should prevail in such specifications.

An example of abbreviation is the coloured net of figure 7. In this case the diferentiation of tokens by means of "colours" wich correspond to the function keys selected, leads to a representation complexity independent of the number of keys in the console.

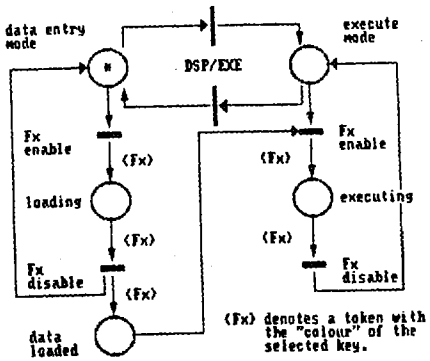

fig. 7 Console logic specification using a coloured net

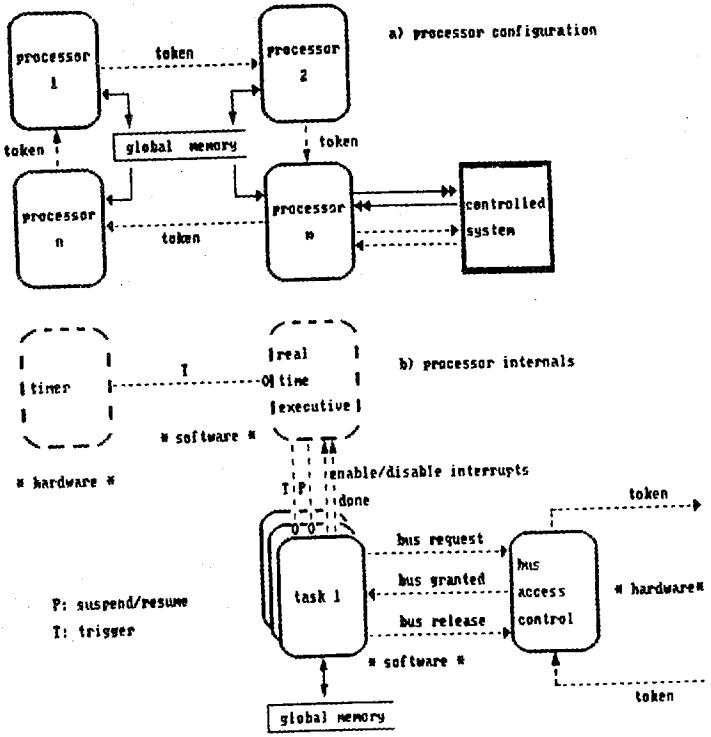

fig. 8 Processor model for a real-time system

Previous example, has shown the need to keep track of several simultaneous events because of the design of the man-machine interface. In figure 8 example, paralelism arises from the existence of multiple processors in a system and multiple tasks within each processor.

To specify the bus access control logic of one processor we could use either state-transition diagrams, figure 9.a, or Petri nets, figure 9.b, but if we want to model all the bus controllers linked to each other, we must resort to Petri nets, as done in figure 10.

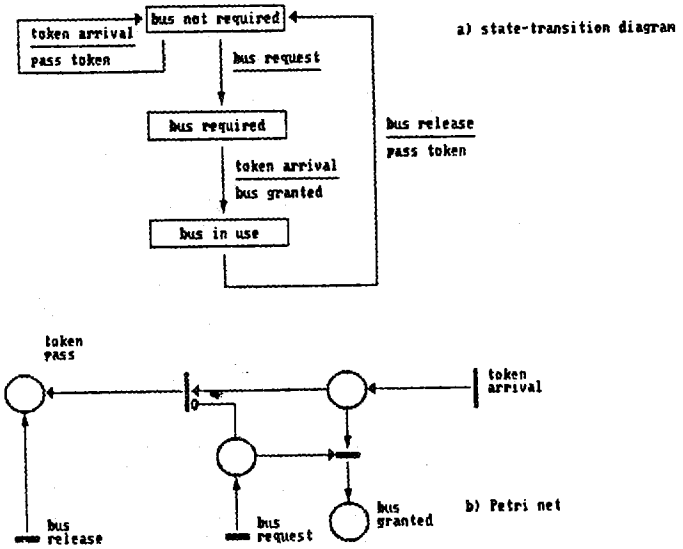

fig. 9 Bus access control specification

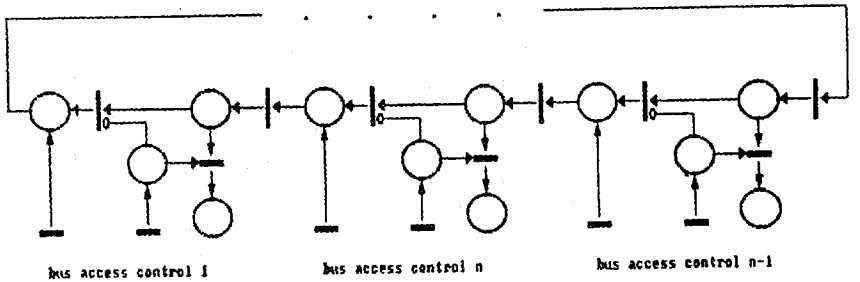

fig. 10 Linking bus controllers

The real-time executive has to deal with tasks runing at diferent priority levels, keeping information about active, preempted and interruptable tasks. The use of prioritized nets, leads to a simple specification, figure 11, of this relatively complex transformation.

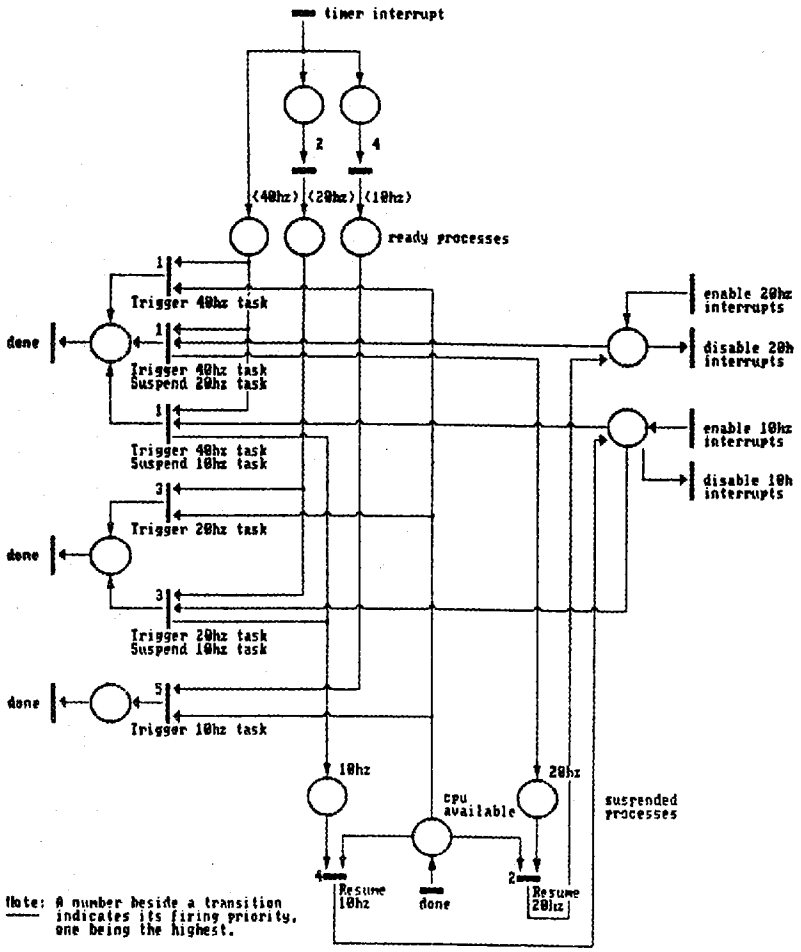

fig. 11 Real-time executive

4. Conclusions

The examples presented have shown that some control transformations are not conveniently and even theoretically specifiable in terms of state-transition diagrams, and how Petri nets can help in some of these situations. The differences between these tools arise from two facts:

- In Petri nets, system state is represented by the distribution (marking) of tokens into the places of the net, and not by the places itself, as is the case with the nodes of the state-transition diagram.

- More powerful primitives allow synchronization to be specified and abbreviations and extensions to nets permit more condensed specifications to be written.

Furthermore the symbolic execution of the transformation schema [1], a topic not considered here, is facilitated if control transformations are specified using the Petri net approach.

References:

1 - ESML: An Extended System Modeling Language Based on the Data Flow Diagram, Bruyn, Jensen, Keskar, Ward, Software Engeneering Notes, Jan 1988.

2 - Structured Development for Real-Time Systems, Ward-Mellow Vol I, II, III. Prentice Hall 1986.

3 - Reseaux de Petri: Theorie et Pratique, Vol I, II. G.W. Brams, Masson 1983.