University of Castilla-La Mancha

A publication of the

Department of Computer Science

New Features of TPAL for the Specification and Analysis of Concurrent Systems

by Juan José Pardo Mateo, Valentín Valero Ruiz, Mª Carmen Ruiz,

Mª Emilia Cambronero.

Technical Report **#DIAB-02-01-25** May 2002

This Report is an extension of the Technical Report DIAB-01-01-14

DEPARTAMENTO DE INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

1 Introduction

Computer systems controlling and carrying out very important and critical tasks are more and more used every day. Any minimum error in these systems can suppose great financial losses but also loss of human lives in some cases.

Then, it has become necessary to define a formal and rigorous framework for the specification and analysis of there before their implementation. Following up these ideas, a great number of researchers have defined specification languages guided toward the specification of concurrent and distributed systems.

Once this formal setting has been defined, tools based on computers supporting some of those formalisms are used to facilitate specification, design and verification tasks. There are a number of tools supporting the different proposals of models and their extensions, among them UPPAAL [BLPY95], CADP, LOLA (Lotos Laboratory)[DIT95].

Efficiency is one of the main characteristics in any tool supporting the specification of concurrent systems, but it will be necessary to face up to systems of great size, whose complexity can make them untackled, and it can even end up being impossible to represent them in the memory of any computer, however big it may be. This is known as the problem of the "State Explosion", and in order to solve it, it is necessary to reduce significantly the specification size or the state graphs that are derived from it.

Our research group *(http://www.info-ab.uclm.es/fmc)* is working on the TPAL tool (*ftp://penelope.info-ab.uclm.es/TPAL*, *http://www.info-ab.uclm.es/fmc/tpaltool/tpaltool.htm*) which provides tool support for the specification of timed and probabilistic concurrent systems. TPAL began to be developed in 1997, and simultaneously to its development the formal framework in which is based has been developed. Dynamic graphs and timed-arc Petri nets are part of this framework. TPAL catches specifications which have been carried out in an algebraic language of the same name, and from them several graphic models are generated, such as dynamic state graphs, timed-arc Petri nets, and, in the future, timed automata. These models will allow us to verify properties of behaviour of the system and to carry out a simulation of the execution of the specified system.

TPAL is available for Linux systems under platform IBM/PC as well as Solaris under platform Sun Sparc. Their operation requirements are scarce; it only needs as additional software: the Xforms graphic libraries. Also, it is advisable to have great quantity of RAM, since the size of the generated graphs can be very large, and they must reside in main memory.

1.1 NEW FEATURES OF TPAL

The current version of TPAL has suffered important modifications from the previous one. The first change has been the user interface, the current one is shown in Figure 1.

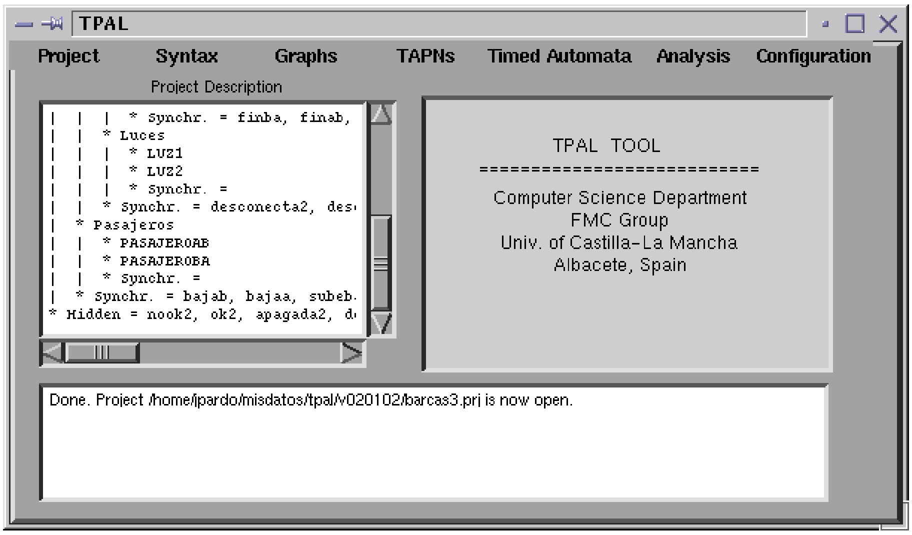


Figure 1: Main Window in TPAL

Other modifications are concerned with the main characteristics and the functionality of TPAL. These characteristics are the following ones:

- It allows the development and edition of specifications in TPAL. Likewise, it allows the development and edition of projects which will allow us to break down the specification of complex systems into several components containing processes.
- Syntactic and semantic analysis of specifications.
- Visualisation of syntactic trees in text mode.
- Translation of the timed specifications into Dynamic State Graphs.
- Simple reduction of the generated graphs.
- Visualisation of graphs in text or graphic mode.
- Simulation of the system, automatic or manually, using the obtained graphs.
- After a simulation, it allows us to save the followed trace for later on to be recovered or studied at length.
- Translation of the specifications into timed-arc Petri nets.

- Visualisation, in text mode, of the obtained timed-arc Petri nets.

In following sections of this document we will proceed to describe the different characteristic of the tool and the objectives we seek to achieve in future.

1.2 Example

In order to illustrate the different characteristics of this tool, we will use the following example, which models the transport way in a river, whose performance is defined as follows.

We considered a river in which there are two boats to cross it in both senses. At the beginning there is a boat in each bank, and each boat only may hold a passenger. If there is a boat when a passenger arrives to a bank, he moves to the opposite bank. In the other hand, if there is not any boat, the clients have a button in each bank of the river, which is associated to a light in the opposite bank. It is lit when pressing the button and it remains lit until the operator of the boat turn off it from the bank where the light is. This will be made when the operator leave in order to assist that request. Several clients can be waiting while nobody appears in the opposite bank, in this case the clients are pressing the button every so often, until they are assisted.

2 Syntactic and semantic analysis

2.1 Syntax of the projects

In this new version, the specifications are organised in projects, each one consists of several processes which can be contained in components. Each project is defined in a file with extension "*.prj*" whose structure is shown in Figure 2.

In this file, first of all we indicate the name of all the processes involved in the project. Once these processes have been specified, each component is defined. These components consist of some of the processes enumerated in the section PROCESSES. For each component it will be necessary to specify the group of actions in which the involved processes synchronise. To conclude the file we include the part corresponding to the whole project, in the section *project*, whose name must coincide with the name of the file .prj, without extension. For the project, it is also necessary to indicate the processes or components involved, specifying the synchronisation actions and he hidden ones. In the file we can include comments, preceding them with double dash.

The project file for the example of the boats is shown in the figure 3.

```
 -- List of processes involved:
 --     EXAMPLE:    Processes P1, P2;
 Processes  ;
 -- List of Components involved in the project specification
 -- This list could be empty:
 --     EXAMPLE: Component C1
 Component P123
 -- Enumeration of processes in the component (at least one):
   Processes  ;
 -- List of actions that must be synchronised for this component
 -- (can be empty)
   Sync     ;
 End_component
 -- Other components, include new component specifications:
 -- Project specification, the name must coincide with that
 -- of this file, without the extension:
 Project ;
 -- List of synchronised actions
   Sync  msg;
 -- List of hidden actions
   Hidden in, out;
End_project
```

Figure 2: Project file

After developing a new project, a directory, whose name is the same of the project but with the extension ".dir.", is created. The files corresponding to the specifications of each process must be placed in this directory. The name of these files must coincide with the corresponding one of the process they specify.

2.2 Syntax of the processes.

The specification of each process will consist of a main process following this syntax:

```
 SPECIFICATION <id_process> [<list_actions>];
 BEHAVIOUR [PROBABILISTIC | NON_DETERMINISTIC]
    <body_process>
    [WHERE <list_process >]
ENDSPEC
```

Where

id_process. This is the identifier of the specification.

list_actions. This is a list that contains all actions used inside the specification.

list_process. This is the specification of all auxiliary processes used in the specification. These are specified according to the following syntax:

```
Processes B1, B2, C1, C2, LUZ1, LUZ2, PASAJEROAB, PASAJEROBA;
Component Barcas
   Processes B1, B2 ;
   Sync     ;
End_component
Component Control
   Processes C1, C2;
   Sync ;
End_Component
Component Luces
   Processes LUZ1, LUZ2;
   Sync ;
End_component
Component ControlBarcas
   Processes Barcas, Control;
   Sync initab,initba,finab,finba;
End_component
Component Pasajeros
   Processes PASAJEROAB, PASAJEROBA;
   Sync;
End_component
Component BarcasControlesLuces
   Processes ControlBarcas, Luces;
   Syn apagada1,apagada2,ok1,ok2,nook1,nook2,desconecta1,desconecta2;
End_component
Project barcas3;
   Processes BarcasControlesLuces, Pasajeros;
   Sync  pulsa1,pulsa2,subeab,subeba,bajaa,bajab;
   Hidden initab,initba,finab,finba,apagada1,ok1,nook1,pulsa1,
          desconecta1,pulsa2,desconecta2,apagada2,ok2,nook2;
End_project
```

Figure 3: Project file for the example of the boats

```
 PROCESS <id_process> [<list_actions>];
 BEHAVIOUR [PROBABILISTIC |NON_DETERMINISTIC]
    <body_process>
ENDPROC
```

body_process is the definition of the behaviour of that process. The definition is made according to the syntax of the algebraic language TPAL, whose formal definition is in [PVCC01].

In order to make a better automatic treatment, some slight modifications in the syntax have been introduced:

- The internal action is represented as i rather than τ.
- The value ∞ is represented by $*$.
- The parallel operator is represented as $[A]$, being A the synchronisation set.
- The external choice operator is represented as "*or*."

Hence, the syntax used for the specification of the bodies of the processes is:

```
<body_process> ::= T
    T::= N | P
    N::= N1 | N2
    N1::= stop | a<t1,t2>;T | i;T | wait(t);T
    N2::= N or N | N[A]N | N\a | X | rec X.N
    P::= [q1]N3 +...+ [qn]N3 | P[A]P | P\a | X | rec X.P
    N3::= N1 | rec X.N1
```

where $t, t1 \in \mathbb{R}_0^+$, $t2 \in \mathbb{R}_0^+ \cup \{*\}$, ($*$ stands for ∞), a $\in$ "list_actions", qi $\in$ (0,1) and A $\subseteq$ list_actions.

The process type (PROBABILISTIC or NON_DETERMINISTIC) can be optionally indicated for each one. If we do not indicate it, in the most cases the tool will be able to determine it.

Besides these slight modifications to the syntax, we impose the following syntactic restrictions to the allowed terms:

The terms should be regular, that is, given a process $\mu x.P$, all appearances of X in P must be prefixed, and neither parallel operator nor hiding operator should appear in the specification of the process P.

Now, we show the specification of the example that we presented above.

```
SPECIFICATION PASAJEROBA
[pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                          initab,initba,vab,vba,finab,finba,subeab,subeba,
                          bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X. pasea<50,*>; (rec Y. ( (subeba<0,0>; bajaa<0,*>; PASAJEROAB1)
                                 or (wait(2); pulsa1<0,0>; Y) ) )

WHERE PROCESS
PASAJEROAB1[pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                    initab,initba,vab,vba,finab,finba,subeab,subeba,
                    bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
    rec X. pasea<5,*>; (rec Y. ( (subeab<0,0>; bajab<0,*>; PASAJEROBA)
                                 or (wait(2); pulsa1<0,0>; Y) ) )
ENDPROC ENDSPEC
```

```
SPECIFICATION PASAJEROAB
[pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                          initab,initba,vab,vba,finab,finba,subeab,subeba,
                          bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
    rec X. pasea<5,10>; (rec Y. ( (subeab<0,0>; bajab<0,*>; PASAJEROBA1)
                                 or  (wait(2); pulsa1<0,0>; Y) ) )

WHERE PROCESS
PASAJEROBA1[pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                    initab,initba,vab,vba,finab,finba,subeab,subeba,
                    bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X. pasea<7,10>; (rec Y. ( (subeba<0,0>; bajaa<0,*>; PASAJEROAB)
                                 or (wait(2); pulsa1<0,0>; Y) ) )
ENDPROC ENDSPEC
```

```
SPECIFICATION C2 [pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                  initab,initba,vab,vba,finab,finba,subeab,subeba,
                  bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X. initba<0,*>; finba<0,*>;X
ENDSPEC

SPECIFICATION C1 [pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                  initab,initba,vab,vba,finab,finba,subeab,subeba,
                  bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
    rec X. initab<0,*>; finab<0,*>;X
ENDSPEC
```

```
SPECIFICATION LUZ1 [pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                    initab,initba,vab,vba,finab,finba,subeab,subeba,
                    bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
  rec X. ( (apagada1<0,*>; ok1<0,*>; X)
           or (desconecta1<0,*>; LUZ1) or (pulsa1<0,*>; LUZ11))
WHERE PROCESS LUZ11[pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
              initab,initba,vab,vba,finab,finba,subeab,subeba,
              bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X. ( (apagada1<0,*>; nook1<0,*>; X)
            or (pulsa1<0,*>; X) or (desconecta1<0,*>; LUZ1) )
ENDPROC ENDSPEC

SPECIFICATION LUZ2 [pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                    initab,initba,vab,vba,finab,finba,subeab,subeba,
                    bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
  rec X. ( ( apagada2<0,*>; ok2<0,*>; X)
    or ( desconecta2<0,*>; X) or (pulsa2<0,*>; LUZ22) )
WHERE PROCESS LUZ22[pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
              initab,initba,vab,vba,finab,finba,subeab,subeba,
              bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X. ( ( apagada2<0,*>; nook2<0,*>; X)
            or ( pulsa2<0,*>; X) or ( desconecta2<0,*>; LUZ2))
ENDPROC ENDSPEC
```

```
SPECIFICATION B1 [pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                  initab,initba,vab,vba,finab,finba,subeab,subeba,
                  bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X.( ( subeab<0,*>; desconecta1<0,*>; initab<0,*>;
             vab<10,15>; finab<0,0>; bajab<0,0>; B22 )
          or ( wait(1); apagada1<0,0>;
         ( ( ok1<0,*>; X) or
               ( nook1<0,*>; desconecta1<0,*>; initab<0,*>;
                 vab<10,15>; finab<0,0>; B22 ) ) ) )
WHERE PROCESS B22[pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
            initab,initba,vab,vba,finab,finba,subeab,subeba,
            bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X.( ( subeba<0,*>; desconecta2<0,*>; initba<0,*>;
             vba<10,15>; finba<0,0>; bajaa<0,0>; B1  )
           or ( wait(1); apagada2<0,0>;
              ( ( ok2<0,0>; X ) or
               ( nook2<0,*>; desconecta2<0,*>; initba<0,*>;
                 vba<10,15>; finba<0,0>; B1 ) ) ) )
ENDPROC ENDSPEC

SPECIFICATION B2 [pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
                  initab,initba,vab,vba,finab,finba,subeab,subeba,
                  bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X.( ( subeba<0,*>; desconecta2<0,*>; initba<0,*>;
             vba<10,15>; finba<0,0>; bajaa<0,0>; B11 )
           or  ( wait(1); apagada2<0,0>;
               (( ok2<0,0>; X ) or
               ( nook2<0,*>; desconecta2<0,*>; initba<0,*>;
                 vba<10,15>; finba<0,0>; B11 ) ) ) )
WHERE

PROCESS B11[pasea,ok1,apagada1,nook1,pulsa1,desconecta1,
            initab,initba,vab,vba,finab,finba,subeab,subeba,
            bajaa,bajab,pulsa2,desconecta2,apagada2,ok2,nook2];
BEHAVIOUR
   rec X.( ( subeab<0,*>; desconecta1<0,*>; initab<0,*>;
             vab<10,15>; finab<0,0>; bajab<0,0>; B2 )
          or ( wait(1); apagada1<0,0>;
             ( ( ok1<0,*>; X) or
               ( nook1<0,*>; desconecta1<0,*>; initab<0,*>;
                 vab<10,15>; finab<0,0>; B2 ) ) )  )
ENDPROC ENDSPEC
```

2.3 Syntactic and Semantic analysis

The syntactic and semantics correctness is made in several phases.

In the first phase it is checked that the specifications follow the syntax of the language; obtaining as a result the first version of the syntactic tree, which is incomplete due to the lack of certain semantic information: the types of the processes and identifiers.

In the second phase, this information is determined (whenever it is possible). However, if the type of the processes or identifiers cannot be detected automatically, an error is generated, and the user must indicate the type explicitly (PROBABILISTIC /

NON_DETERMINISTIC).

After concluding the syntactic and semantic analysis we have a syntactic tree for each process of the specification. These trees have the information of the main operator of the process in the root node, and each child is the subtree corresponding to each argument of that operator.

Among the information stored for each operator we have the type of the process (NON_DETERMINISTIC or PROBABILISTIC), the description (operator) and the arguments.

In Figure 4 we can observe the definition of the data structures which are used for building the graph.

```
struct NODO {
  char Tipo;             /* node type (P:prob.; N:non-deterministic) */
  char Desc;             /* Description of higher level operador */
  struct Argumentos Arg; /*  Arguments */
  struct NODO *Hijos[MAXPALT];  /* Children Nodes */
  int numhijos;                 /* Number of children */
};

struct Argumentos{
/* prefix: */
     char Nombre[TAMANOMBRES];   /* Action name */
     unsigned int t1, t2;        /* Time of timed prefix */
/* Wait: */
     unsigned int t1;            /* Time of wait */
/* Parallel: */
     struct LISTAR c_sinc;       /* Set of synchronisation actions */
/* Hidden */
     char Nombre [TAMANOMBRE];   /* Hidden action */
/* Identifier */
     char Nombre [TAMANOMBRE];   /* Identifier name */
/* Recursion */
     char Nombre [TAMANOMBRE];   /* Identifier name */
/* Probabilistic choice */
     float p [MAXPALT];          /* Probabilities*/
};
```

Figure 4: Information of syntactic tree

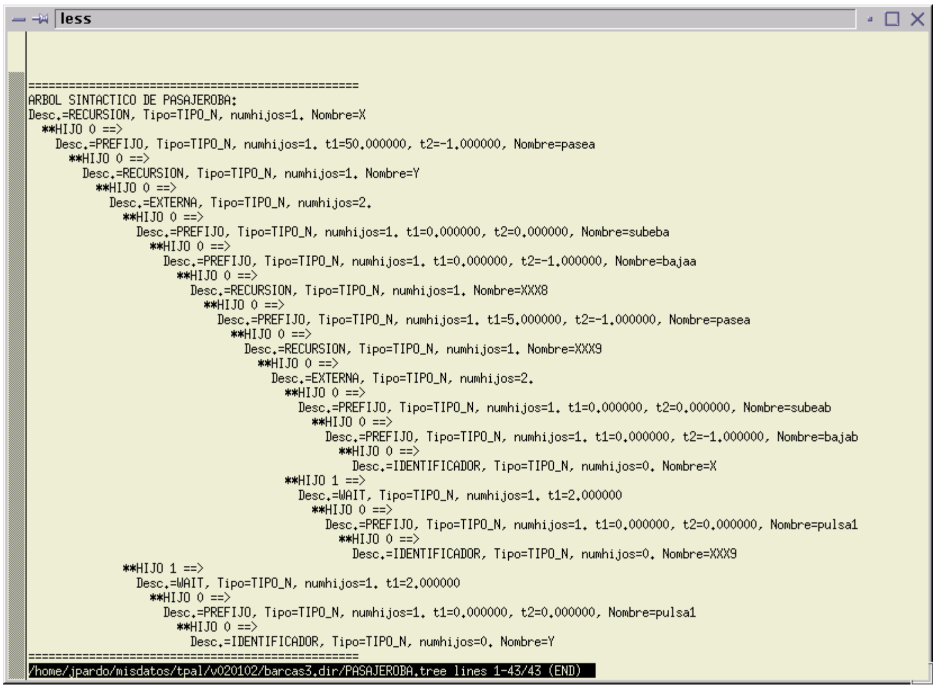
If we focus again our attention on the problem of the boats for instance we obtain the tree shown in Fig 5 for the process PASAJEROBA.

3 Graphs

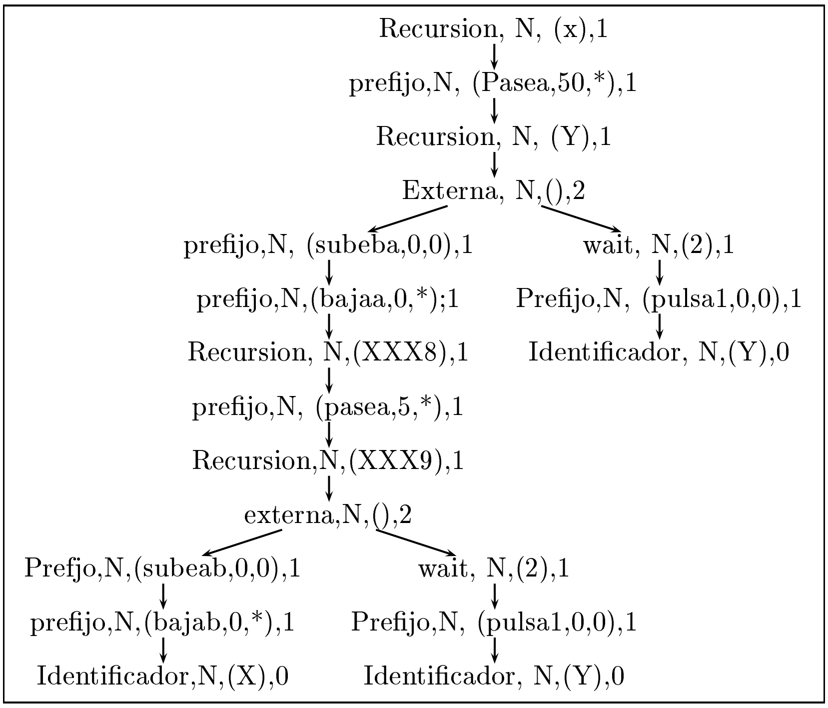
Starting from the syntactic trees, we may generate the dynamic state graphs, although the current version only deals with timed processes, but we hope to conclude the implementation for probabilistic processes soon.

This translation follows the formal translation described in [PVCC01].

To optimise the administration of memory in the machine on which the tool is executed



(a) Text view shown by the tool



(b) Graphic view

Figure 5: Syntactic tree generated for the process PASAJEROBA

and to lessen the problem of state explosion, instead of building a graph for the whole system, as it was made in the previous version of the tool, a modular graph is built, which consists of a graph for each process in the specification.

In this modular definition, we have each separate graph, as well as some additional information about the whole system, such as synchronisation actions, hidden actions and a global clock.

3.1 Browser and Simulation

In this new version both functions has been joined, hence when the graph is shown the simulation can be accomplished.

As the graphs are shown, several windows appears, each one corresponding to each graph, where nodes are pictured in three colors:

Red There is just a red-colored node. This node is the active one in each moment.

Yellow These are the nodes that have some adjacent nodes that cannot be painted due to the lack of space in the window.

white The rest of nodes.

Some parameters can be configured, such as the size of the nodes and their labels. In this new version of the tool the labels of the transitions can be shown or hidden.

In Figure 6 we can see the windows generated by the tool in order to show the graphs generated for the problem of the boats.

3.1.1 Simulation

The simulation window appears next to the windows showing the graphs. This simulation window is shown in Figure 7.

In the top left-hand corner the values of each clock appear, while in the bottom the trace which is being executed is shown.

Some simulation parameters can also be configured (Fig 8), among them we can highlight the simulation type, if time-locks must be avoided, the simulation speed, etc.

If the *automatic simulation* is chosen, the system chooses randomly the transition to be executed among the enabled ones at each moment. If the option "*Avoid time locks*" has been selected, the selection of the action to be executed is made trying to avoid the deadlock of the system by the expiration of the actions.

If the option of manual simulation is chosen, the user of the tool controls the whole evolution of the system.

To indicate the time elapsing, the user must click the Delay button in the simulation window (see Figure 7), after this, he indicates the time he wants to elapse. This value has an upper bound in function of the urgent actions that are enabled.

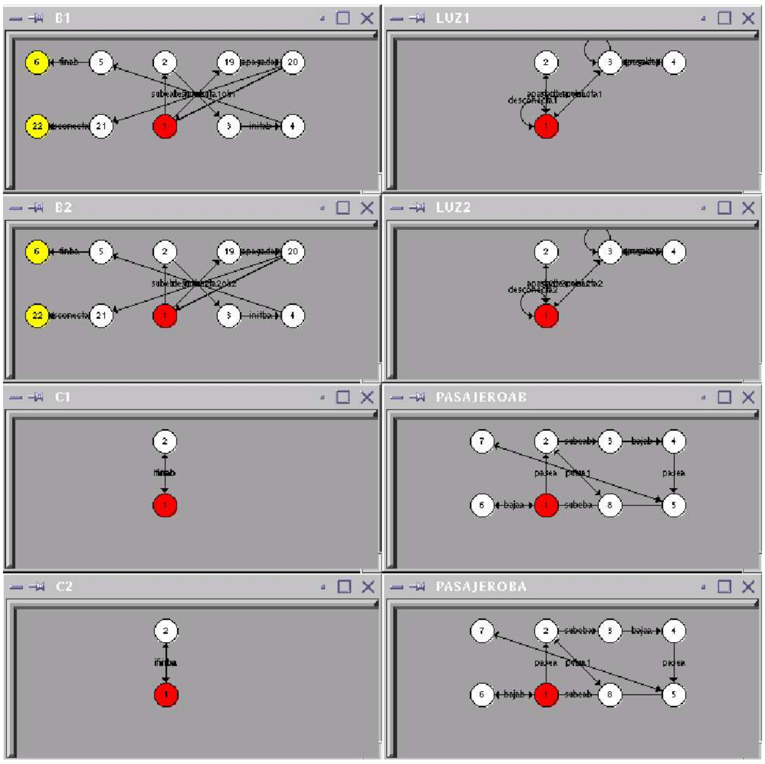


Figure 6: Browser of graphs

At every instant, all transitions enabled appear red colored. We just need to click on the mouse in order to execute the actions.

If this transition is labelled with an action belonging to the synchronisation group with another process, the transition is colored in blue and the system waits until the complementary action with which it must synchronise is selected.

When the graphs are shown, we can click on any edge or node with the right button of the mouse to obtain information about it, (Figures 9(a) and 9(b)).

3.2 Traces

As a new functionality associated with the simulation of the system we have the possibility of saving in a file the trace of the actions executed until a certain moment. The stored information is composed of the name of the executed action and the instant of time in which it has been executed.

Later on, our intention is that they can be imported to reproduce an experiment.

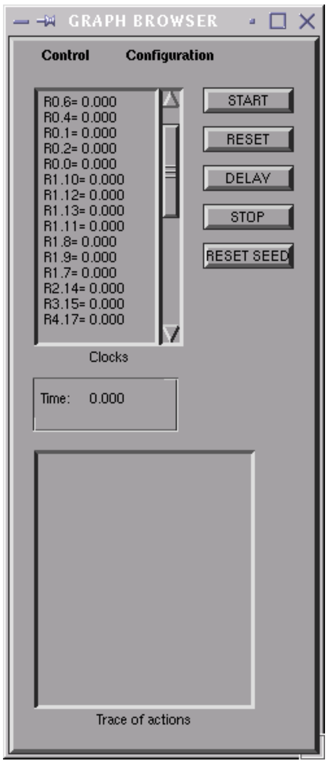


Figure 7: Simulation window

4 Timed-arc Petri Nets

One of the important innovations of the tool is the possibility of translating TPAL specifications into Timed-arc Petri nets [BLT90, Wal83, Han93, VdFC99, AN01], in which each token has associated an age, and the arcs going from a place to a transition are labelled with time interval. Thereby, the token that is going to be used to fire a transition must have an age belonging to the interval that labels the arc that connects this place with the transition.

The theoretical support of this translation can be found in [VPC02], where the main features of the language TPAL and the particular model of timed-arc Petri nets used are shown.

The net result of this translation is a unique timed-arc Petri net for the whole system. Currently, the tool only allows the visualisation of the Petri nets in text mode but we hope to conclude the component for the graphic visualisation soon.

At present the tool implements a small kernel of timed-arc Petri nets that even allows its simulation, although due to the lack of an interface this feature has not been used yet.

Figure 10 shows a partial view of the text file corresponding to the timed-arc Petri net obtained from the example of the boats.

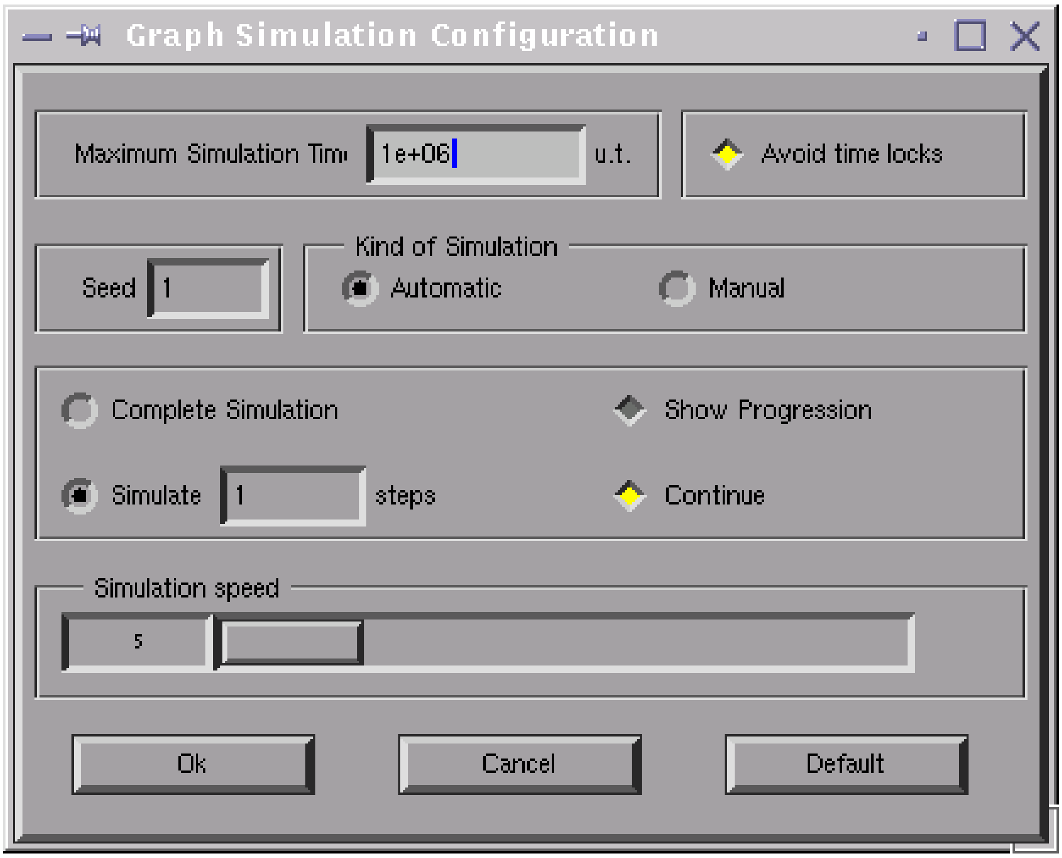
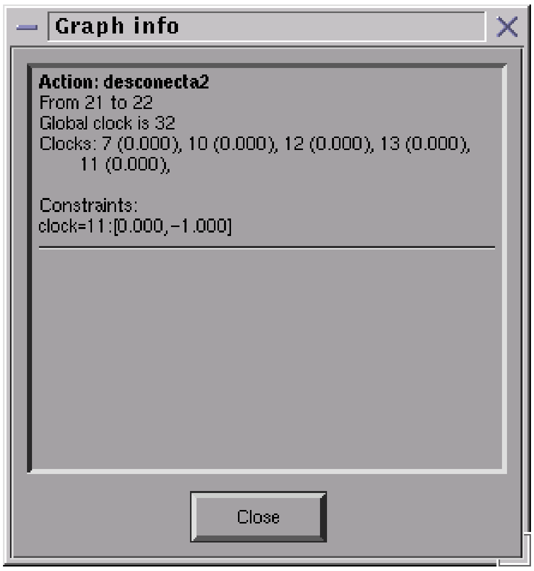


Figure 8: Table of configuration of the simulation

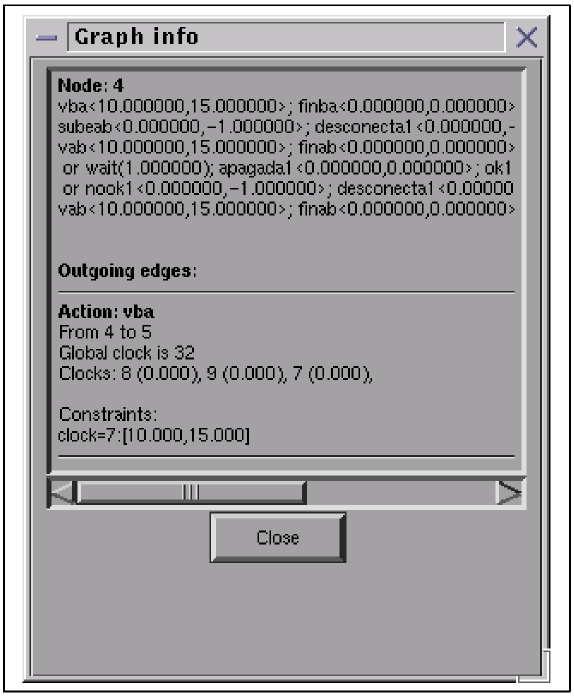
5 Future works

The functionality of the tool has been increased through its successive versions. However, there are still several features that we consider interesting and which are part of our future works:

- Implementation of a display of Petri nets which allows a static visualisation and the simulation of the generated nets. For this, it will be necessary the connection of this display with the kernel of the simulation, which is already implemented.
- Implementation of algorithms for verification of properties by means of timed-arc Petri nets.
- Direct translation from the obtained graphs, or from specifications, into timed automata in a first step and later on to generalise them to probabilistic and timed automata.
- Analysis and verification of the systems by means of model-checking.
- Development and edition of timed-arc Petri nets directly using a graphic environment.



(a) Edge Information



(b) Node Information

Figure 9: Information of graph elements

References

[AN01] Parosh A. Abdulla and Aletta Nylen. Timed Petri Nets an BQOs. In *Proc 22nd International Conference on Theory and Application of Petri Nets.*, volume 2075 of *Lecture Notes in Computer Science*, pages 53–70, 2001.

[BLPY95] J. Bengtsson, K. Larsen, P. Peterson, and Wang Yi. Uppaal.- a tool suite for automatic verfication of real-time systems. In *Proc of 4th Dimacs Workshop on Verification and Control of Hybrid Systems*, New Jersey, October 1995.

[BLT90] T. Bolognesi, F. Lucidi, and S Trigila. From timed Petri nets to timed LOTOS. In *Proceedings of Tenth International IFIP WG6.1 Simposium on Protocol Specification Testing and Verification*. North-Hollan, 1990.

[DIT95] DIT UPM. *LOtos LAboratory. User Manual (Version 3R6)*, February 1995.

[Han93] Hans-Michael Hanisch. Analysis of place/transition nets with timed-arcs and its aplication to batch process control. In *Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 282–299, 1993.

[PVCC01] J. Pardo, V Valero, F Cuartero, and D. Cazorla. Automatic Translation of TPAL specification into Dynamic State Graphs. In *Proc of APSEC'01*. IEEE Computer Society Press, 2001.

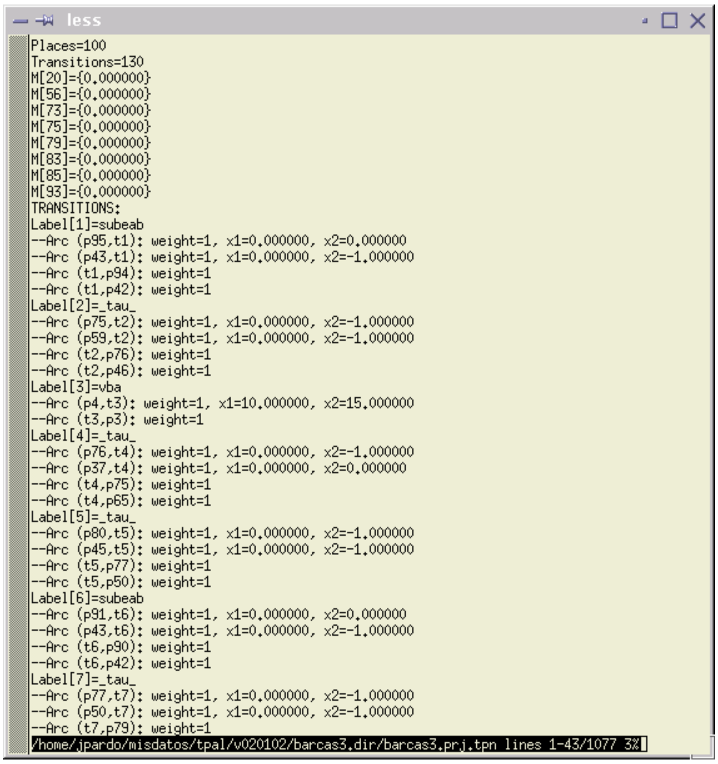


Figure 10: Visualisation of the Petri net generated

[VdFC99] V. Valero, D. de Frutos, and F Cuartero. Decidability of the Strict Reachability for Timed-Arc Petri Nets. In *Proc 8th Int. Workshop on Petri Nets and Performance Models PNPM'99*, pages 188–196, 1999.

[VPC02] Valentín Valero, Juan J. Pardo, and Fernando Cuartero. Translating TPAL Specifications Into Timed-Arc Petri Nets. In *Proccedings of ICTAPN'02*, 2002. to appear.

[Wal83] B. Walter. Timed Petri-Nets for Modelling and Analysing Protocols with Real-Time Characteristics. In *Proc 3rd IFIP Workshop on Protocol Specification, Testing and Verification*. North-Holland, 1983.