University of Castilla-La Mancha

A publication of the

Department of Computer Science

Herramienta de Ayuda al Diseño y Evaluación de Sistemas Concurrentes.

by

Juan José Pardo Mateo, Valentín Valero Ruiz.

Technical Report **#DIAB-01-01-14** December 2000

This Report is an extension of the Technical Report DIAB-99-01-05

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 Introducción

Cada día más, existen sistemas informáticos que controlan y realizan tareas muy importantes y críticas. Cualquier mínimo error en estos sistemas puede suponer una gran perdida económica y lo que puede resultar más importante, en vidas humanas.

Esto hace que sea necesario definir un marco formal y riguroso donde estos sistemas puedan ser especificados, analizados y verificados antes de su implementación. Estas ideas han llevado a una gran cantidad de investigadores a a realizar la definición de lenguajes de especificación y concretamente para sistemas concurrentes. En la literatura podemos encontrar algunas extensiones temporizadas y probabilsticas de estos lenguajes (ver [6]).

Una vez definido este marco, y para facilitar la tarea, se utilizan herramientas que facilitan la especificación y la verificación de sistemas concurrentes. Entre ellas podemos destacas UPPAAL [5, 2], CADP [4] y LOLA (LOtos LAboratory) [3].

TPAL es una herramienta para la verificación y simulación de sistemas concurrentes, especificados mediante un Algebra de procesos temporizada y probabilstica. Estas especificaciones son traducidas a un tipo de grafos temporizados y probabilísticos, que son parecidos a los automatas temporizados (ver [1]).

La principal ventaja de esta herramienta es que pueden escribirse especificaciones directamente usando el lenguaje algebrico, siemdo generado automticamente el grafo correspondiente. Es obvio que el problema de la explosin de estados aparece al modelar sistemas no triviales. Con la finalidad de solucionar este problema,actualmente, solo se aplican algunos algoritmos muy simples de reducción de grafos, pero en un futuro muy prximo se realizara una extensin de la herramienta de modo que permita la especificación modular.

La versión actual , que puede ser obtenida de *http://penelope.info-ab.uclm.es/TPAL*, posee las siguientes caracteristicas:

- Permite la creación y edición de especificaciones en el lenguaje TPAL.
- Análisis sintactico y semntico de las especificaciones.
- Generación de los Grafos de Estados Dinámicos correspondientes a las especificaciones (Solo para procesos temporizados).
- Reducción simple de los grafos generados.
- Visor del grafo y utilidad de simulación del sistema usando el grafo.
- Configuración de diversos parametros del visualizador y de la simulación.
- Calculo de cotas de tiempo para el instante de llegada a un nodo.

2 Sintaxis de las especificaciones

Las especificaciones aceptadas por TPAL están compuestas por varios especificaciones de procesos , donde uno de ellos es el principal, que se adecuará a la sigiente estructura:

SPECIFICATION < identifier > [< list_of_actions >];
BEHAVIOUR [PROBABILISTIC | NON_DETERMINISTIC]
< body_process >
[WHERE < list_of_processes >]
ENDSPEC

donde

identifier es el nombre del proceso principal,

lista de acciones es el conjunto de acciones utilizadas en la especificación

process es la especificación en si misma

list_of_processes es una lista de especificaciones que corresponden a procesos auxiliares utilizados.Estos se adecuan a la siguiente sintaxis

PROCESS < process_identifier > [<list_of_actions >];
BEHAVIOUR [PROBABILISTIC | NON_DETERMINISTIC]
< body_process >
ENDPROC

El body_process se realizan de acuerdo a la siguiente notación BNF:

< body_process> ::= T

T ::= N | P

N::=N1 | N2

N1 ::= stop | a$< t_1, t_2 >$;T | i;T | wait(t);T

N2 ::= N or N | N [A] N | N\a | X | rec X.N

P ::= [q_1]N3 + ... + [q_n]N3 | P [A] P | P\a | X | rec X.P

N3 ::= N1 | rec X.N1

donde $t_1 \in \mathbb{R}$, $t_2 \in \mathbb{R}^+ \cup \{*\}$, $q_i \in (0, 1)$

El signo $*$ indicará el valor infinito.

3 Construcción de la herramienta

3.1 Entorno de la trabajo

Actualmente se está construyendo la herramienta para que funcione sobre una maquina PC con el sistema operativo linux en entorno gráfico X-Windows. Para ellos se utiliza el lenguaje de programación C y las librerías Xforms para la construcción del entorno gráfico. La elección de estas librerías se debió a las grandes posibilidades que ofrecía, a la facilidad de uso de las mismas, y a la existencia de una herramienta gráfica como es Fdesign que permite la creación de entornos de forma fácil y cómoda.

En una fase posterior, se realizará una versión para el sistema operativo Solaris de Sun, e incluso se podría plantear la posibilidad de obtener una versión para Windows Microsoft.

A continuación realizaremos una descripción de cada uno de los componentes de esta herramienta, indicando estructuras de datos utilizadas.

3.2 Componentes

3.2.1 Analizador

Es el primer y uno de los más importantes componentes de la herramienta, teniendo como principal objetivo el comprobar que las especificaciones introducidas se ajustan a la sintaxis del lenguaje. Como resultado se obtendrá un árbol sintáctico de la expresión introducida.

Este analizador está compuesto de tres partes interrelacionadas y complementarias las cuales pasamos a describir a continuación.

Scanner o Analizador Léxico Su finalidad es la de leer el fichero fuente para obtener los tokens que representan cada uno de los símbolos del lenguaje. Para ello realiza una lectura del fichero carácter a carácter.

Parser o Analizador Sintáctico Es el encargado de controlar todo el proceso de análisis de una especificación. Este análisis es realizado en dos fases, de modo que en la primera se construye el árbol sintáctico correspondiente al proceso especificado dejando sin resolver algunas determinaciones de tipos, las cuales son resueltas en la segunda fase, a partir de los tipos de los nodos hijos.

Análisis Semántico El análisis semántico es realizado a la par que el sintáctico y de hecho está dirigido por éste. Durante el análisis sintáctico se construye el árbol sintáctico del proceso analizado, el cual es un árbol binario donde el nodo raiz contiene la información del operador principal y cada una de las ramas son los subárboles correspondientes a sus dos argumentos. Para los operadores con un solo argumento como puede ser la acción prefijada se tendrá que el árbol asociado solamente tendrá la rama izquierda. La definición en C de las estructuras que implementan estos árboles sintácticos será la siguiente.

```
/* Argumentos operadores */
struct Argumentos
{
 /* stop, div, interna urgente, or  : No tienen */
 /* Prefijo: */
     char Nombre[TAMANOMBRES];
     unsigned int t1, t2;
 /* Wait: */
     unsigned int t1;
 /* Paralelo: */
     struct LISTAR *c_sinc;
 /* Ocultamiento: */
     char Nombre[TAMANOMBRES];
 /* Identificador: */
     char Nombre[TAMANOMBRES];
 /* Recursion: */
    char Nombre[TAMANOMBRES];
 /* Eleccion Probabilistica: */
    float p[MAXPALT];
};
```

```
struct NODO {
  char Tipo;                    /* tipo de nodo */
  char Desc;                    /* Descripcion de operador de mas alto nivel */
  struct Argumentos Arg;        /* Argumentos */
  struct NODO *Hijos[MAXPALT];  /* Nodos hijos */
  int numhijos;                 /* Numero de hijos */
};
```

3.2.2 Generador del grafo

Tras la realización del análisis de un expresión, y una vez comprobada su corrección, se construye el grafo de transición de estados asociado a ese proceso y que es obtenido mediante la aplicación de la definición mostrada en [6], tomando como partida el árbol sintáctico generado por el parser.

3.3 Estructuras de datos manejadas

Al construir el grafo, debemos recoger toda la información que en la definición formal se recogía, para lo cual definimos las siguientes estructuras de datos.

- **Información sobre relojes**

  Definimos un array con un determinado número máximo de elementos (en la actualidad es de 100) que será el número máximo de relojes que se podrán utilizar, y en el cual se almacena el valor de cada uno de los relojes. Esta estructura es definida global, de modo que se utiliza la misma para la generación de los grafos de todos los subprocesos de una especificación.

- **Información de arcos**

  La información que necesitamos almacenar sobre los arcos es bastante diversa y abundante. Concretamente debemos tener información sobre los siguientes aspectos:

  - Identificador de la acción que debe ejecutarse para realizar la transición.
  - Restricciones que deben cumplirse para que pueda ser ejecutada la transición.
  - Relojes que deben ser actualizados al realizarse la transición.

  Adicionalmente y por motivos de implementación se tendrá también información sobre el nodo origen y destino del arco.

  Para recoger todos estos datos la estructura definida en C será:

```
struct INFO_ARCO{
   struct INFO_NODO *origen, *destino;
   char *accion;
   struct LISTA_RESTR *constraints;
   struct CONJUNTO_RELOJES *relojes_sinc;
   struct INFO_ARCO *siguiente;
};
```

  Donde el campo constraints es una lista de nodos del tipo mostrado a continuación, que indica las diferentes condiciones de habilitación de ese arco.

```
struct LISTA_RESTR{
  int reloj;    /* numero del reloj */
  float t1, t2; /* Restricciones (t2 igual a -1 equivale a infinito) */
  struct LISTA_RESTR *siguiente;
};
```

- **Información de nodos**

  En los nodos solamente información sobre el proceso que lo etiqueta, así como un puntero a una lista de arcos que de este nodo parten.

```
struct INFO_NODO{
   unsigned int numero;
   struct TERMINO_NODO *termino;
   struct INFO_ARCO *primero;
   struct INFO_NODO *siguiente;
};


struct TERMINO_NODO{
   unsigned int referencias;
   BOOLEAN EsParalelo;
   union p {
        struct NODO *t;
        struct Paralelo{
           struct TERMINO_NODO *t1;
           struct LISTAR *conj_sinc;
           struct TERMINO_NODO *t2;
        } Paralelo;
   } p;
};
```

- **Estructura del grafo**

  A la hora de almacenar en memoria el grafo, y debido a la gran cantidad de nodos y arcos que puede tener un grafo, hemos optado por una versión basada en listas de adyacencias, de modo que cada nodo tendrá asociada una lista con los arcos que salen de él. Igualmente los nodos aparecerán en una lista enlazada.

  Además de esta información tendremos almacenado el número del reloj que hace las funciones de reloj principal así como una lista de todos los relojes utilizados por el grafo.

  Asi pues la estructura de C que contiene la información de un grafo será:

```
struct GRAFO_DINAMICO{
   struct INFO_NODO *cabeza;
   unsigned   int num_nodos;
   unsigned   int num_arcos;
   struct   CONJUNTO_RELOJES *relojes;
   unsigned int reloj_global;
  };
```

4 La interfaz de usuario de la herramienta

Pasemos a continuación a realizar una breve descripción del funcionamiento de esta herramienta.

Para iniciar la aplicación ejecutaremos el comando **tpal**, tras lo cual aparecerá la ventana principal de la misma, la cual podemos ver en la figura 1.

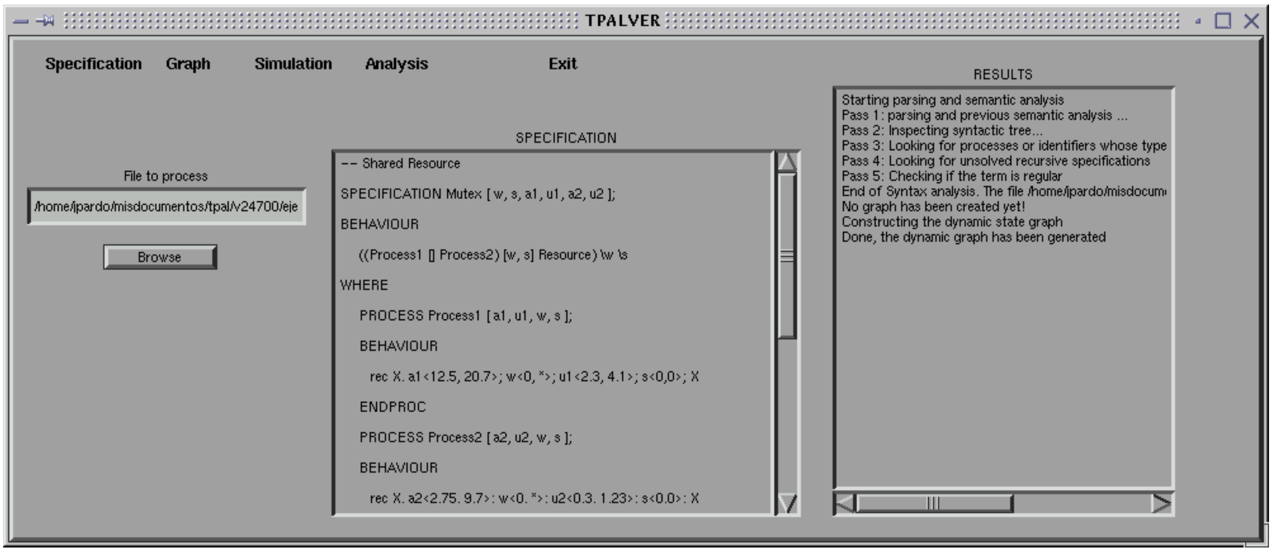

Figura 1: Ventana principal de la Herramienta

4.1 Especificaciones

4.1.1 Apertura

En primer lugar lo que debemos hacer es indicar el fichero que contenga la especificación que deseamos procesar para ellos tenemos tres alternativas.

1. Podemos escribir directamente el nombre del mismo en el recuadro **File to process**. Al terminar de introducir el nombre deberemos pulsar enter. En este caso será necesario introducir el path completo del fichero.
2. Si pulsamos sobre el botón browse que aparece en la ventana nos aparecerá un selector de ficheros mediante el cual podremos desplazarnos por el árbol de directorios hasta seleccionar el fichero deseado.
3. Podemos seleccionar la opción Open File del menu Specification con lo cual se mostrará tamb ién el selector de ficheros.

Tras seleccionar el fichero deseado en el recuadro Specification aparecerá ésta.

4.1.2 Edición

Para la edición de la especificación sobre la que se está trabajando debemos seleccionar la opción Edit del menú Specification. Esto lanzará un nuevo proceso con el editor joe, que nos permitirá modificar la especificación. Tras salvar los cambios y salir del editor se actualizará el recuadro de la especificación. (ver figura 1.)

4.1.3 Análisis sintáctico

Es la primera fase del proceso de una especificación, y comprobará que la especificación se adecua a la sintaxis. Para ello debemos seleccionar la opción Syntax Analysis del menu Specification.

Esta deberá realizarse obligatoriamenete al principio.

4.2 Grafos

Una vez realizado el análisis sintáctico de una especificación y si ésta es correcta, podemos realizar el grafo correspondiente atendiendo a la definición formal de los mismos. Para ello seleccionaremos la opción Generation del menú Graph.

Una vez generado el grafo, podemos visualizarlo, mostrando el fichero de texto o el grafo de forma gráfica. Para ello seleccionaremos la opcion correspondiente del menú graph. Esta visualización, en caso de grafos muy grandes será parcial, mostrandose en color amarillo aquellos nodos cuyos nodos adyacentes no pueden ser mostrados. En estos casos nosotros podemos pulsar sobre uno de estos nodos, de modo que pase a ser el nodo actual y entonces se redibujar d enuevo el grafo. El nodo actual sera mostrado en color rojo.

4.3 Simulación

También tenemos la posibilidad de realizar una simulación, bien de forma manual o de forma automática. En este último caso podemos habilitar la opción de "Avoid timelocks", lo que intentará evitar un deadlock jpor motivos de lla expiracion de tiempos.

Durante una simulación tendremos transiciones que se muestran en color rojo, que indican que están habilitadas para ser ejecutadas y en negro aquellas que no lo están. Para ejecutar una de las transiciones que estén habilitadas deberemos pinchar sobre ellas con el botón izquierdo del ratón. Si pulsamos sobre cualquier transición o nodo con el botón derecho, se abrirá una nueva ventana mostrando información sobre el objeto seleccionado.

Durante la simulación, además de la ejecución de transiciones podemos indicar el paso del tiempo, para ello pulsamos sobre el botón Delay de la ventana de simulación, tras lo cual se nos solicitará el tiempo que dejamos pasar. Este paso del tiempo podrá habilitar algunas transiciones así como deshabilitar otras que estuviesen ya habilitadas.

4.3.1 Configuración

Antes de iniciar una simulación podemos definir varios parametros del funcionamiento de la misma. Para ello seleccionamos la opción configuration del menú Simulation, tras lo cual se mostrará la ventana de esta configuración, la cual podemos verla en la figura 2

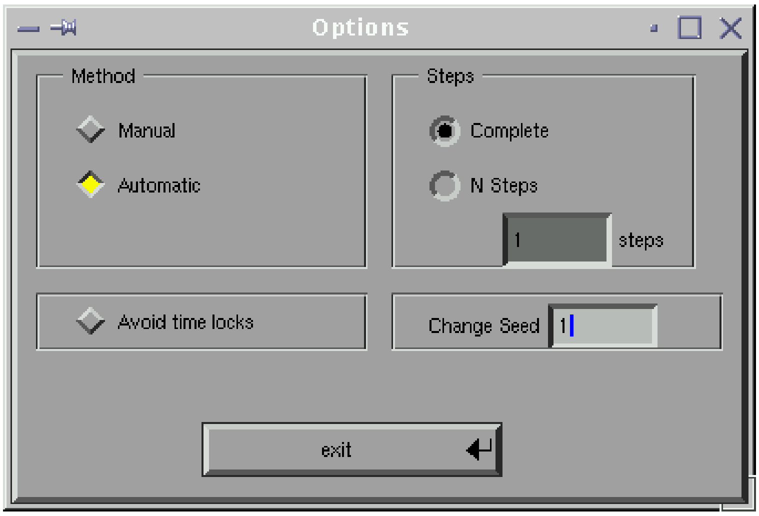

Figura 2: Ventana de configuración de la Simulación

En ella podemos indicar si la simulación se hará de forma manual o automática, si se realizará completa o solamente N pasos.

4.3.2 Reset

Seleccionando la opción Re-Inicialization del menú Simulation volveremos el estado inicial de la simulación para volver a comenzar ésta.

4.4 Análisis

Una vez creado el grafo, podemos realizar sobre el mismo el análisis de algunas propiedades, como es el calculo de una cota superior e inferior para el tiempo de llegada a un nodo dado. Para ello debemos indicar el camino seguido hasta llegar a ese nodo.

5 Trabajos futuros

Los trabajos que se plantean para incrementar la utilidad de esta herramienta son los siguientes:

- Modificación de la visualización de los grafos para que permita mostrar el grafo completo.
- Implementación de algunos algoritmos para el estudio de propiedades interesantes sobre el sistema.
- Obtención de versiones del programa para otros sistemas operativos tales como Sun-Solaris y Windows.
- Ampliación para el tratamiento de especificaciones con operadores probabilísticos.

Referencias

[1] R. Alur and D. L. Dill, *Automata-theoretic verification of real-time systems.*, Formal Methods for Real-Time Computing (Constance Heitmeyer and Dino Mandrioli, eds.), Wiley, 1996, pp. 55–82.

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

[3] DIT UPM, *Lotos laboratory. user manual (version 3r6)*, February 1995.

[4] J.C. Fernandez, H. Garabel, A Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, *Cadp a protocol validation and verification toolbox*, Proc. of the 8th Conference on Computer-Aided Verification (New Jersey (USA)), 1996, pp. 437–440.

[5] Larsen K., P. Petterson, and W. Yi, *Diagnostic model-checking for real-time systems*, Proc. of 4th Dimacs Workshop on Verificiation and control of Hybrid Systems. (New Jersey), October 1995.

[6] Juan J. Pardo, V. Valero, and Cuartero F., *A dynamic state graph for a timed process algebra*, Proceedings of SNPD 2000, May.