Centro de Investigación y de Estudios Avanzados del IPN

Departamento de Ingeniería Eléctrica
Sección Computación

Desarrollo de un Sistema Gráfico para Análisis, Diseño y Monitoreo de Tareas de Tiempo Real

Tesis que presenta

Miguel Ángel Fajardo Ortiz

Para obtener el grado de:

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica Opción Computación

Asesor de Tesis:

Dr. Pedro Mejía Alvarez

México, D.F. Octubre de 2003.

Resumen

El problema de planificar un conjunto de tareas de tiempo real sobre sistemas uniprocesadores es considerado como un problema altamente complejo y de muchas variantes. Esto se debe al hecho de que este sistema debe ejecutarse siempre de forma predecible y cumplir con plazos de respuesta predefinidos. Un sistema de tiempo real se compone de un conjunto de tareas que se ejecutan de forma concurrente. Por lo tanto, las tareas de tiempo real compiten por el CPU, y por los recursos del sistema.

El problema del cumplimiento de los plazos de respuestas de las tareas resulta difícil debido a que no siempre se obtienen o se modelan con precisión los parámetros temporales. Para cumplir con sus requerimientos temporales (plazos de respuesta), las tareas de un sistema de tiempo real deben obtener con precisión sus tiempos de cómputo, y aplicar un algoritmo de planificación que les permita ejecutarse de forma predecible y siguiendo determinadas prioridades de ejecución. El algoritmo de planificación se encarga de asignar prioridades de ejecución a las tareas de forma tal que ninguna de estas pierda sus plazos de respuesta.

Desafortunadamente, hoy en dia existen pocas herramientas de diseño que permitan validar y monitorizar los parámetros temporales de un sistema de tiempo real. Por este motivo, en esta tesis nos propusimos el desarrollo de una herramienta que facilite el diseño y el análisis de tareas de tiempo real, y que permita hacer un seguimiento de su ejecución de forma gráfica. En la herramienta desarrollada, se han incluido los esquemas de planificación con *prioridades fijas: Rate Monotonic (RM), Deadline Monotonic (DM)* y con *prioridades dinámicas: Earliest Deadline First (EDF), Least Laxity First (LLF).*

Para el desarrollo de la herramienta nos hemos enfocado en conjuntos de *tareas periódicas, desalojables (preemptive)*, para ambientes de un solo procesador. La herramienta posee dos modos de funcionamiento: Modo estático y, modo dinámico. En el modo estático, el usuario de la herramienta proporciona los parámetros temporales de cada tarea, mientras que para el modo de trabajo dinámico, los datos son proporcionados por un archivo generado por un Micro-Kernel de tiempo real.

Abstract

The problem of scheduling a set of real-time tasks on uniprocessor systems, is considered as a highly complex problem of many variants. This problem is related to the fact that this system must always execute in a predictable manner, while meeting timing constraints.

A real-time system is composed of a number of processes (or tasks) that execute concurrently, hence there is a contention for CPU and other computational resources. The problem of meeting deadlines is complex, because timing parameters of real-time task are not always obtained accurately. In order to solve this problem, the system designer must obtain timing requirements of tasks (computation times) with a high level of accuracy, and must apply scheduling algorithms to execute tasks following a predictable order. Scheduling algorithms, allow real-time tasks to follow a predefined execution order, so that they always meet their deadlines.

Unfortunately, there exist very few design tools capable of modeling and simulating the temporal behavior of real-time systems. For this reason, in this thesis work, we propose the design of a graphical design tool capable to simulate and validate the temporal execution of a set of real-time tasks.

Within this graphical tool we implemented scheduling schemes with fixed priorities such as Rate Monotonic (RM), and Deadline Monotonic (DM), and with dynamic priorities such as Earliest Deadline First (EDF) and Least Laxity First (LLF). The models used for simulation in our graphical tool, focused in periodic preemptive tasks for a single processor environment. The tool has embedded two modes of operation: an static mode and a dynamic mode. In the static mode, the timing parameters of each real-time task are provided by the user. In

the dynamic mode the data is provided by a file generated by a real-time micro-kernel.

Agradecimientos

A Dios, por permitirme terminar satisfactoriamente la maestría.

Al Dr. Pedro Mejía Alvarez, le agradezco por darme la oportunidad de participar en su proyecto, por su preocupación y entusiasmo para que esta tesis llegará a buen fin. También a los doctores Dr. Francisco Rodríguez Henríquez y Dr. Adriano de Luca Pennacchia por sus consejos en la escritura de este documento.

A los doctores de la sección de computación por compartir su entusiasmo por el conocimiento y ser partícipes de mi vida acádemica, en especial a los doctores: Dr. Feliú Sagols, Dr. Luis Gerardo de la Fraga, Dr. Jorge Buenabad, Dr. Adriano de Luca, Dr. Arturo Díaz, Dr. José Oscar Olmedo, Dra. Xiaoou Li y al Dr. Guillermo Morales.

A mi madre por el apoyo incondicional que me dio en todo momento, y enseñarme a tener fortaleza de continuar hacia adelante, sobreponiendome siempre a las adversidades que la vida presenta. Gracias por guiarme por el camino de la educación.

A mis hermanos y a mi sobrina Monserrat, gracias por su cariño.

A Sofia Reza por el apoyo administrativo a lo largo de la maestria.

Y a toda la gente que directa o indirectamente me ayudó en algún momento para poder estar donde estoy ahora.

Índice General

Resumen i

Abstract iii

Agradecimientos v

Índice General x

Índice de Figuras xiii

Índice de Tablas xv

Lista de Símbolos xviii

1 Introducción **1**

1.1 Introducción 1

1.2 Objetivos de la Tesis 3

1.2.1 Objetivo General 3

1.2.2 Objetivos Específicos 3

1.3 Organización del Documento 4

2 Introducción a los Sistemas de Tiempo Real **5**

2.1 Definición de Sistemas de Tiempo Real 5

2.2 Elementos de un Sistema de Tiempo Real 6

2.3 Estados de los Procesos 8

2.4 Tipos de Restricciones de las Tareas de Tiempo Real 10

2.4.1 Restricciones de Tiempo 11

2.4.2 Restricciones de Precedencia 13

2.4.3 Restricciones de Recursos 14

2.5 Definición del Problema de Planificación 15

2.6 Paradigmas de Planificación 15
2.6.1 Planificación basada en Prioridades Estáticas 16
2.6.2 Planificación basada en Prioridades Dinámicas 17
2.7 Métricas para el Evaluación de Funcionamiento 21
2.8 Planificación de Tareas Periódicas 22
2.9 Factor de Utilización 23
2.10 Algoritmo de Planificación Cíclico 24
2.11 Algoritmos de Planificación Estáticos 26
2.11.1 Rate Monotonic (RM) 26
2.11.2 Condición de Planificabilidad Suficiente y Necesaria 30
2.11.3 Deadline Monotonic (DM) 33
2.12 Algoritmos de Planificación Dinámicos 33
2.12.1 Earliest Deadline First (EDF) 33
2.12.2 Least Laxity First (LLF) 34

3 Trabajos Relacionados **37**
3.1 ASSERTS 38
3.1.1 Especificación de componentes del Hardware y el Kernel 38
3.1.2 Especificación de algoritmos de planificación 39
3.1.3 Especificación de las tareas 39
3.1.4 Activación 41
3.1.5 Interfaz Gráfica de Usuario 41
3.2 PERTS 41
3.2.1 Modelo de Sistemas de Tiempo Real en PERTS 41
3.2.2 Vista general de PERTS 43
3.2.3 Grafos de las Tareas y los Recursos 43
3.2.4 Jerarquía de Planificación 44
3.3 STRESS 44
3.3.1 Ejecución del simulador 46
3.3.2 Redes, Buzones, Nodos y Procesadores 47
3.3.3 Tareas 47

3.4 GHOST 48
3.4.1 Modelo de la Máquina Virtual 49
3.4.2 Algoritmos de planificación en GHOST 50
3.4.3 Generador de carga 51
3.4.4 Módulo de salida 52
3.5 ARTISST 53
3.5.1 Características 53
3.5.2 Modularidad 54
3.5.3 Modelo de Tareas en ARTISST 55
3.5.4 Estados de las Tareas 56
3.6 AFTER 56
3.6.1 Implementación 56
3.6.2 Unidad de Monitoreo 57
3.6.3 Unidad de Filtro 58
3.6.4 Unidad de Extracción 58
3.6.5 Unidad de Análisis 59
3.6.6 Predicción 59

4 Herramienta de Simulación: PlataTR **61**
4.1 Motivación 61
4.2 Modelo del Sistema 62
4.3 Diagrama General de Bloques 64
4.4 Estructura de la Interfaz Gráfica de Usuario 69
4.4.1 Ventana Principal 71
4.4.2 Ventanas de Diálogo 74
4.5 Metodología para la Planificación de un Conjunto de Tareas 77

5 Implementación y Ejemplos **81**
5.1 Implementación de los eventos de tiempo real 81
5.2 Clases implementadas en la herramienta 82
5.2.1 Clase CPlataView 82

5.2.2 Clase CGraphics 85
5.2.3 Clase CClassArray 86
5.2.4 Clase CScheduler 87
5.2.5 Clase CReal 89
5.2.6 Clase CPlataSlider 91
5.2.7 Clase CMisDatos 91
5.2.8 Clase CGridCtrlDlg 91
5.2.9 Clase CGridCtrl 92
5.2.10 Clase CGridCellBase 92
5.3 Ejemplos 92
5.3.1 Ejemplo de una Aplicación de Tiempo Real 92
5.3.2 Ejemplo de una Aplicación generada por el Micro-Kernel 99
5.3.3 Ejemplos con Parámetros Temporales Conocidos 103

6 Conclusiones y Trabajo Futuro **119**
6.1 Conclusiones 119
6.2 Trabajo Futuro 120

A Manual de Usuario **123**
A.1 Requerimientos 123
A.2 Estructura del directorio 124
A.3 Iniciando PlataTR 125
A.4 Manejo de PlataTR 126
A.4.1 Captura de un Conjunto de Tareas 126
A.4.2 Editar 128
A.4.3 Ejecutando un algoritmo de planificación 129
A.4.4 Detener y Reanudar la Ejecución 129

Bibliografía **131**

Índice de Figuras

2.1 Elementos de un Sistema de Tiempo Real. 6
2.2 Diagrama de estados de un proceso. 10
2.3 Parámetros de una Tarea de Tiempo Real 12
2.4 Relación de precedencia para cinco tareas. 14
2.5 Clasificación de los algoritmos de planificación para sistemas de tiempo real. 18
2.6 Planificación cíclica 25
2.7 Conjunto de tareas que no satisface la Ecuación 2.7 y por lo tanto no es planificable 28
2.8 Conjunto de tareas que satisface 2.7 y por tanto es planificable 29
2.9 Conjunto de tareas que no satisface 2.7. Sin embargo, es planificable 30
2.10 Puntos de Planificación 32
2.11 Planificación de las tareas de la tabla 2.8 bajo EDF. 34
2.12 Planificación de tareas bajo el algoritmo de planificación LLF. 35

3.1 Un modelo de referencia para sistemas de tiempo real en PERTS 42
3.2 Programa sencillo utilizando STRESS 46
3.3 Interfaz gráfica de STRESS 47
3.4 Interfaz gráfica de GHOST 49
3.5 Estructura general de la herramienta GHOST 50
3.6 Ejemplo de un módulo de especificación de salida definido por el usuario en GHOST 52
3.7 Modelo de simulación de ARTISST 53
3.8 Arquitectura básica para la evaluación del sistema simulado 54

3.9 Componentes de AFTER 57
3.10 Datos procesados por AFTER 58
3.11 Interfaz gráfica: Modificación de parámetros del sistema de tiempo real . . . 60

4.1 Tiempos relativos para las tareas.. 64
4.2 Estructura lógica de PlataTR 65
4.3 Ventana de Windows para almacenar un archivo de tareas en PlataTR. . . . 68
4.4 Arquitectura documento-vista de PlataTR 70
4.5 Ventana principal de PlataTR 71
4.6 Opciones de menú de PlataTR 72
4.7 Parámetros de entrada de un conjunto nuevo de tareas 76
4.8 Tabla de edición de las tareas 76
4.9 Seleccionar el algoritmo de planificación de se desea aplicar al conjunto de tareas 77
4.10 Metodología de diseño y análisis de un conjunto de tareas de tiempo real . . 78

5.1 Jerarquía de Clases de PlataTR 83
5.2 Elementos visuales de las Clases de Objetos de PlataTR 84
5.3 Añadiendo un nuevo algoritmo de planificación en PlataTR desde Visual C++ 6.0 87
5.4 Propiedades del Botón de Opción para el nuevo algoritmo que se añade a PlataTR 88
5.5 Sistema de tiempo real que regula temperatura y nivel de líquido en un Tanque. 93
5.6 Proceso de planificación 93
5.7 Primera asignación de parámetros temporales 95
5.8 Planificación con RM falla. 96
5.9 Primera asignación de parámetros temporales 97
5.10 Planificación con RM, el conjunto es planificable. 97
5.11 Planificación con EDF, con los parámetros asignados en la Figura 5.7. 98
5.12 Muestra de las tareas que PlataTR ha recibido del Micro-Kernel de Tiempo Real 101
5.13 Visualización de los eventos en el intervalo de tiempo [0,30] 102

5.14 Visualización de los eventos en el intervalo de tiempo [30,60] 102
5.15 Visualización de los eventos en el intervalo de tiempo [89,119] 103
5.16 Especificaciones temporales ejemplo 1 104
5.17 Secuencia temporal del ejemplo 1 bajo el algoritmo RM, intervalo [0,30] . . . 105
5.18 Ssecuencia temporal del ejemplo 1 bajo el algoritmo RM, intervalo [30,60] . . 105
5.19 Secuencia temporal del ejemplo 1 bajo el algoritmo RM, intervalo [60,90] . . 106
5.20 Secuencia temporal del ejemplo 1 bajo el algoritmo EDF, intervalo [0,30] . . 107
5.21 Secuencia temporal del ejemplo 1 bajo el algoritmo EDF, intervalo [30,60] . . 107
5.22 Secuencia temporal del ejemplo 1 bajo el algoritmo EDF, intervalo [60,90] . 108
5.23 Especificaciones temporales para el ejemplo 2 109
5.24 Secuencia temporal del ejemplo 2 bajo el algoritmo RM, intervalo [0,30] . . . 110
5.25 Secuencia temporal del ejemplo 2 bajo el algoritmo RM, intervalo [60,90] . . 110
5.26 Especificaciones temporales para el ejemplo 3 111
5.27 Secuencia temporal del ejemplo 3 bajo el algoritmo RM, intervalo [0,30] . . . 112
5.28 Secuencia temporal del ejemplo 3 bajo el algoritmo RM, intervalo [30,60] . . 112
5.29 Especificaciones temporales para el ejemplo 4 113
5.30 Secuencia gráfica ejemplo 4 bajo el algoritmo EDF, rango [0,30] 114
5.31 Especificaciones temporales para el ejemplo 5 115
5.32 Secuencia gráfica del ejemplo 5 bajo el algoritmo LLF, para el intervalo [0,30] 116
5.33 Secuencia gráfica ejemplo 5 bajo el algoritmo LLF, para el intervalo [30,60] . 116
5.34 Secuencia gráfica ejemplo 5 bajo el algoritmo LLF, para el intervalo [60,90] . 117

A.1 Iniciando PlataTR desde el Explorador de Windows. 126
A.2 Captura de parámetros temporales para crear un nuevo conjunto de tareas . 127
A.3 Ventana de mensaje de error 128
A.4 Ventana de edición de tareas. 129
A.5 Algoritmos de planificación de PlataTR 130

Índice de Tablas

2.1 Funciones de costo para evaluar los algoritmos de planificación 22
2.2 Conjunto de tareas periódicas 25
2.3 Utilización mínima garantizada para n tareas 27
2.4 Conjunto de tareas, la utilización total cumple con la condición 2.7 29
2.5 Conjunto de tareas armónico, es planificable 29
2.6 Conjunto de Tareas que no satisface 2.7 31
2.7 Carga en el instante t 32
2.8 Conjunto de tareas, cuya utilización es del 82% 34

3.1 Macro-instrucciones de ASSERTS. 40
3.2 Funciones básicas disponibles en GHOST 51

4.1 Parámetros para las tareas de PlataTR. 64
4.2 Formato de archivo de tareas de tiempo real creado desde PlataTR. 66

5.1 Función utilizada para introducir un nuevo algoritmo de planificación 90
5.2 Tareas generadas por el Micro-Kernel de Tiempo Real 100
5.3 Eventos generados por el Micro-Kernel de Tiempo Real 100
5.4 Prioridad asignada a las tareas del ejemplo 1 por Rate Monotonic 104
5.5 Prioridad asignada en base a su holgura en cada instante de tiempo 115

Lista de Símbolos

τ_i	La i-ésima tarea.
a_i	Tiempo de activación de la tarea (τ_i).
C_i	Tiempo de cómputo de la tarea (τ_i).
D_i	Plazo de respuesta de la tarea (τ_i).
f_i	Tiempo de finalización de la tarea (τ_i).
L_i	Latencia de la tarea (τ_i), $L_i = f_i - D_i$.
N_{e_i}	Número de ejecuciones de la tarea (τ_i) en el cíclo principal, $N_{e_i} = T_M/T_i$.
N_{T_S}	Número de cíclos secundarios en el cíclo principal de la planificación cíclica, $N_{T_s} = T_M/m$.
s_i	Tiempo de inicio de la tarea (τ_i).
T_i	Período de la tarea (τ_i).
T_M	Ciclo principal en la planificación cíclica.
T_S	Ciclo secundario en la planificación cíclica.

Capítulo 1

Introducción

1.1 Introducción

Los recientes avances en la tecnología de Ingeniería de Software han permitido disponer de herramientas para el diseño y prototipado de sistemas de software completos. Desafortunadamente, estas herramientas fueron diseñadas para software de propósito general y no son útiles para modelar o simular sistemas de tiempo real. En un sistema de tiempo real, un conjunto de tareas concurrentes compiten por el procesador, y por otros recursos de cómputo. Su ejecución es de forma concurrente y su principal objetivo consiste en cumplir con sus respectivos plazos de respuesta. Un sistema de tiempo real es un sistema de cómputo, el cual atiende eventos de un ambiente externo, que cuenta con restricciones de tiempos. Ejemplos de este tipo de sistemas, los encontramos en aplicaciones industriales de robótica, multimedia, sistemas de cómputo médico, sistemas de aviónica, o sistemas de cómputo móvil, por mencionar sólo unos cuantos.

En dichas aplicaciones es necesario medir los tiempos de cómputo de cada uno de los procesos (tareas) a fin de verificar si el conjunto de procesos, ejecutandose concurrentemente en uno (o varios) procesadores, cumplen con sus especificaciones temporales. La medición de estos tiempos de cómputo no es tarea fácil, ya que en una aplicación industrial, una tarea de tiempo real puede estar realizando procesamiento tan complejo como:

- lectura y escritura de dispositivos análogico/digitales,
- lectura de dispositivos de entrada/salida,
- lectura y control de sensores y actuadores,
- uso de la red para envío de imagenes, película, texto o gráficos,
- procesamiento de información compleja (p. ej. transformadas de Fourier, solución de sistemas ecuaciones complejas, o encripción o desencripción de información),
- acceso a bases de datos.

En todos estos casos, la medición de su correspondiente tiempo de cómputo puede llegar a ser una tarea difícil. Esto se debe a que muchas de estas acciones son de caracter impredecible y en ocasiones es posible medir con precisión su tiempo de cómputo. Este es uno de los principales problemas a los que se enfrenta un diseñador de un sistema de tiempo real. A pesar de estos problemas, el diseñador debe ser capaz de obtener un tiempo de cómputo, promedio o máximo de ejecución de estos procesos a fin de analizar su comportamiento temporal.

Tradicionalmente en el diseño de los sistemas de tiempo real, primero se desarrolla el software de los procesos de la aplicación, y después se validan los requerimientos temporales mediante técnicas de análisis de planificabilidad o mediante simulaciones. Estos enfoques consumen mucho tiempo y en muchas ocasiones son muy proclives a errores. Los sistemas de tiempo real desarrollados con estas técnicas, resultan difíciles de mantener y extender, ya que alguna pequeña modificación en el software o en la plataforma de hardware, produce efectos impredecibles en el comportamiento temporal del sistema.

La ausencia de herramientas que permitan llevar a cabo un diseño formal y sistemático para construir y validar un sistema de tiempo real, llevan a la construcción de sistemas de tiempo real con errores en su funcionamiento y en su comportamiento temporal. Es por esto que en esta tesis nos planteamos el desarrollo de una herramienta gráfica, que permita modelar y analizar un sistema de tiempo real.

En esta herramienta se plantea la construcción de distintos esquemas de planificación para sistemas de tiempo real. El modelo de tareas de tiempo real se enfoca a tareas estrictas,

periódicas y desalojables, sin recursos compartidos. Los métodos de planificación desarrollados en la herramienta son de prioridades fijas y de prioridades dinámicas. En particular implementaremos los algoritmos RM (Rate Monotonic), DM (Deadline Monotonic), EDF (Earliest Deadline First) y LLF (Least Laxity First).

1.2 Objetivos de la Tesis

1.2.1 Objetivo General

El objetivo principal de este trabajo de tesis es desarrollar una herramienta de simulación que permita verificar, validar, y monitorear un conjunto de tareas concurrentes de tiempo real, mediante una interfaz gráfica.

1.2.2 Objetivos Específicos

Para llevar a cabo el objetivo general, el trabajo se ha dividido en los siguientes objetivos específicos.

1. Analizar e implementar las políticas de planificación para sistemas de tiempo real críticos, en particular las políticas de asignación con prioridades estáticas (Rate Monotonic y Deadline Monotonic) y dinámicas (Earliest Deadline First y Least Laxity First).

2. Proporcionar las herramientas y funciones necesarias para almacenar y obtener los parámetros de las tareas. Para esto, se requieren algunos elementos principales utilizados en los sistemas de software gráfico como son:

   a) **Cajas de diálogo**, cuya función es permitir la entrada, consulta y manipulación de los parámetros temporales.

   b) **Mapas de bits**, donde se visualice la simulación de la ejecución de tareas ordenadas mediante un algoritmo de planificación seleccionado previamente.

   c) **Barras de desplazamiento**, que nos permitan desplazar la ejecución de las tareas hacia la izquierda o hacia la derecha de la ventana de visualización.

3. Integrar los elementos de software anteriormente descritos, en una Interfaz Gráfica de Usuario que facilite el diseño de sistemas de tiempo real.

4. Visualización de las tareas mediante gráficas de Gantt[1].

5. Realizar el análisis del comportamiento de un conjunto de tareas de tiempo real.

6. Realizar simulaciones del comportamiento de conjuntos de tareas generadas externamente, por algún kernel de tiempo real.

1.3 Organización del Documento

La presente tesis está estructurada en seis capítulos y un apéndice.

* En el capítulo 2 se introduce el modelo de tareas, el factor de utilización del procesador y los algoritmos de planificación basados en prioridades estáticas y dinámicas.

* En el capítulo 3 se hace una revisión general de algunas herramientas de simulación para el análisis y monitorización de tareas de tiempo real. En particular en este capítulo, se revisan las siguientes herramientas: ASSERTS, PERTS, STRESS, GHOST, ARTISST y AFTER.

* En el capítulo 4 se presenta la herramienta que desarrollamos, a la que hemos denominado Plata-TR (**Pla**nificador de **Ta**reas de **T**iempo **R**eal), se da una descripción general de la interfaz gráfica.

* En el capítulo 5 se describen los experimentos de simulación llevados acabo para analizar el comportamiento de un determinado conjunto de tareas, bajo los algoritmos implementados en la herramienta.

* El capítulo 6 contiene las conclusiones del trabajo realizado y se da un bosquejo de las posibles líneas de trabajo a futuro.

* Finalmente, se presenta el apéndice A, el cual contiene el Manual de Usuario de PlataTR.

[1]Gráfica de Gantt. Es un diagrama o gráfica de barras que se usa para representar la ejecución de tareas en determinado período de tiempo.

Capítulo 2

Introducción a los Sistemas de Tiempo Real

En este capítulo se definen los conceptos más importantes de los sistemas de tiempo real, los cuales serán utilizados en esta tesis. Así mismo, se enumeran los diferentes tipos de restricciones que pueden específicarse sobre las tareas de tiempo real, de las cuales la más importante para este estudio son las restricciones de tiempo.

De la misma forma, se define el problema de planificación para sistemas de tiempo real, así como la clasificación de los algoritmos de planificación. Los algoritmos de planificación se clasifican en estáticos y dinámicos. Los algoritmos de planificación presentados en esta tesis son RM (Rate Monotonic), DM (Deadline Monotonic), para el caso *estático* y EDF (Earliest Deadline First) y LLF (Least Laxity First) para el caso *dinámico*. Se introducen ejemplos donde se explica la ejecución de conjuntos de tareas planificados bajo estos algoritmos.

2.1 Definición de Sistemas de Tiempo Real

Los *sistemas de tiempo real* se definen como aquellos sistemas en los que su correcto funcionamiento no sólo depende de los resultados lógicos, sino también del momento en que se

producen dichos resultados [2]. Esta interpretación se debe a que estos sistemas interactúan con un ambiente físico a controlar. Sus entradas reflejan eventos ocurridos en su ambiente, y sus salidas se traducen en efectos sobre dicho ambiente. Los resultados deben producirse en los momentos en que aún tengan válidez dentro del ambiente a controlar. Por esta razón, los sistemas de tiempo real controlan un ambiente que tiene restricciones de tiempo bien definidas, es por ello, que se vuelven más complejos y por tanto demandan una alta confiabilidad, con resultados correctos, predecibles y a tiempo.

Actualmente, los sistemas de tiempo real abarcan un amplio rango de aplicaciones, como son los sistemas de control de procesos, sistemas médicos, sistemas de manufactura, sistemas de telefonía móvil, sistemas multimedia, robótica, por nombrar solo unos cuantos.

2.2 Elementos de un Sistema de Tiempo Real

Un sistema de tiempo real consiste principalmente de computadoras, y elementos externos con los cuales el software de la computadora debe interactuar simultáneamente. En la Figura 2.1 se muestran los elementos generales de un sistema de tiempo real donde el objetivo principal es controlar un ambiente. En dicha Figura se distinguen los siguientes elementos:

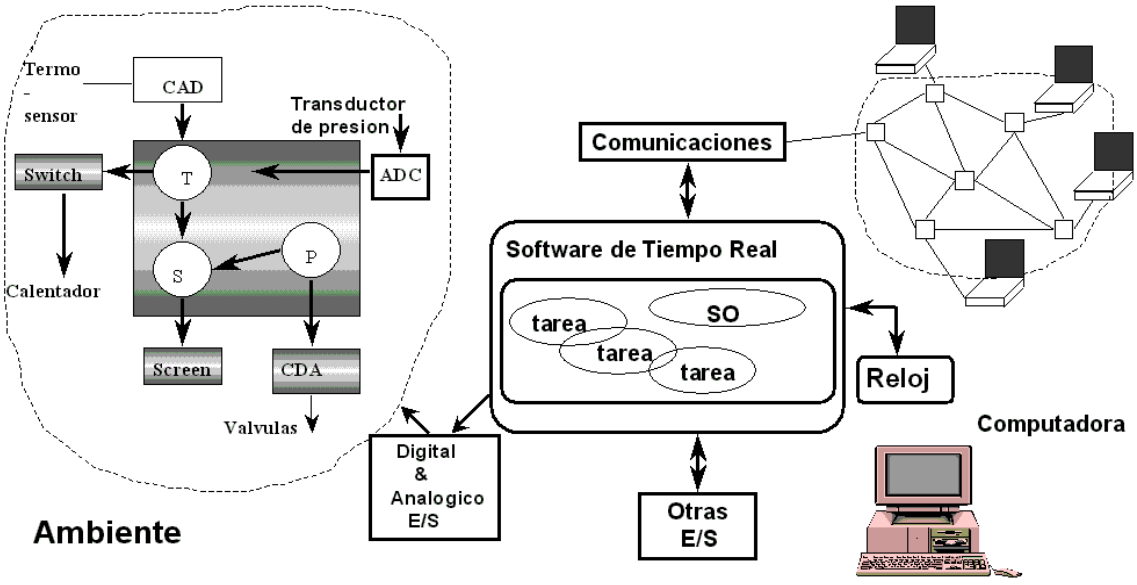

Figura 2.1: Elementos de un Sistema de Tiempo Real.

- **Ambiente**.

  El término ambiente de la Figura 2.1 se refiere al sistema controlado. Por ejemplo, un motor, un sistema de manufactura, un robot, o un avión, etc. El estado del ambiente (entorno físico) es supervizado por los sensores y puede cambiar por los actuadores.

- **Convertidores**.

  Los convertidores análogico-digital convierten las señales generadas por el ambiente (analógicas) a una serie de datos que la computadora interpreta (digitales).

- **Reloj de Tiempo Real**

  El reloj de tiempo real permite al sistema contar el tiempo en que se ejecutan acciones. De la misma forma, el reloj de tiempo real permite al sistema configurar los tiempos para la planificacion de las tareas. Mediante el reloj de tiempo real es posible configurar interrupciones para controlar dispositivos externos, recibir y sincronizar señales de comunicacion, y monitorear el cumplimiento de los requerimientos temporales de las tareas del sistema.

  Sin el reloj de tiempo real no sería posible configurar los tiempos de ejecución de las tareas de tiempo real, y por tanto la planificación del sistema, ni tampoco sería posible saber si las tareas cumplen con sus restricciones temporales.

  En una aplicacion que involucra a varias computadoras es necesario que los relojes de tiempo real se encuentren sincronizados, a fin de que todos ellos lleven la misma cuenta de tiempo.

- **Software de Tiempo Real**

  El software de tiempo real esta compuesto de un sistema operativo (o kernel) y de tareas las cuales son planificadas por el kernel. La estructura del kernel esta compuesta de manejadores de tiempo, de tareas, de memoria, de dispositivos, y de todos los recursos del sistema de cómputo.

  Las tareas del sistema (o procesos) son las entidades de software que permiten controlar el medio ambiente. Cada tarea es un procedimiento de software que se ejecuta de

forma continua, sin embargo, la ejecucion concurrente de las tareas es controlada por el manejador de procesos del kernel.

Existe otro software dentro del sistema, como son las librerías, los *drivers* de dispositivos o los controladores de comunicaciones.

- **Comunicaciones**

  En el sistema de tiempo real pueden existir distintas computadoras que interactúan entre sí. Entre ellas existe un medio físico de comunicación (hardware) y un protocolo de comunicaciones (software) que les permite enlazarse, compartir información y sincronizar su ejecución.

  Mediante la comunicación es posible compartir recursos de hardware (por ejemplo, dispositivos de entrada y salida), y mejorar la eficiencia de aplicaciones mediante la distribución del cómputo, y lograr mayor rapidez de ejecución.

- **Otras E/S (entradas y salidas)**

  Un sistema de tiempo real tiene principalmente como entrada el comportamiento del sistema físico controlado. Sin embargo, existen otras entradas y salidas principalmente aquellas con las que interactúan con el usuario, como son el teclado, el ratón, y otros dispositivos de interacción con el usuario.

2.3 Estados de los Procesos

Un *proceso* se define como un programa en ejecución, tal que su ejecución procede de manera secuencial. En cualquier instante, solo una instrucción se estará ejecutando por proceso en el CPU. Un proceso esta representado por su código, sus datos y su *stack*. El código del proceso es ejecutado en el CPU en forma concurrente con otros procesos, lo cual quiere decir que el CPU es repartido entre varios procesos. Al ejecutar a varios procesos en forma concurrente es necesario saber que cantidad máxima de tiempo de CPU (Quantum de tiempo) se le asigna a cada proceso. Al finalizar este Quantum el planificador asignará el CPU al proceso con mayor prioridad en el sistema. Al procedimiento de cambio de CPU se le conoce como *cambio de contexto*. EL procedimiento de cambio de contexto es el siguiente:

- Interrumpir la ejecución del proceso en ejecución.
- Guardar el lugar de ejecución mediante el contador del programa, o (*Instruction Pointer: IP*) del proceso, sus datos (*Data Segment, DS*) y su stack (*Stack Segment: SS, BP y SP*).
- Buscar el siguiente proceso con mayor prioridad en el sistema.
- Asignar el CPU al proceso con mayor prioridad.

El detalle de la ejecución de los procesos se guarda en una estructura de datos conocida como PCB (*Process Control Block*). En esta estructura de datos se guarda, el estado del proceso, la dirección y tamaño asignado de su código, datos y stack, los recursos que tiene asignados, y posibles manejadores de excepción asignados al proceso. Debemos considerar que, un programa por sí sólo no es un proceso; un programa es una entidad *pasiva*, como el contenido de un archivo guardado en disco, mientras que un proceso es una entidad *activa*, con el contador de programa especificando la siguiente instrucción que se ejecutará, y un conjunto de recursos asociados. Aunque podría haber dos procesos asociados, al mismo programa, de todas maneras se toman como dos secuencias en ejecución distintas. Por ejemplo, varios usuarios podrían estar ejecutando copias del programa de correo, o un mismo usuario podrá invocar muchas copias del programa editor. Cada una de éstas es un proceso aparte y, aunque las secciones de texto sean equivalentes, las secciones de datos variarán. También es común tener un proceso que engendra muchos procesos durante su ejecución.

A medida que un proceso se ejecuta, cambia de *estado*. El estado de un proceso está definido en parte por la actividad actual de ese proceso. Cada proceso puede estar en uno de los siguientes estados:

- **Nuevo (New)**: El proceso esta en disco y no se le han asignado recursos. En este estado, el proceso no puede ejecutarse.
- **En ejecución (Running)**: Se le ha asignado el CPU y estará en ejecución hasta que se termine su *Quantum* de tiempo.
- **En espera (Waiting)**: El proceso está esperando que ocurra algún evento (como la

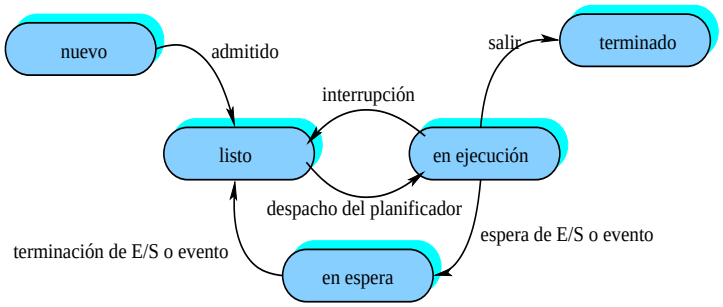

Figura 2.2: Diagrama de estados de un proceso.

terminación de una operación de entrada o salida o la recepción de una señal), o que transcurra un lapso de tiempo.

- **Listo (Ready)**: El proceso esta listo para ejecución y esperando a que se le asigne el procesador.

- **Terminado (Terminated)**: El proceso terminó su ejecución. Se le retiran los recursos y no puede competir mas por tiempo de ejecución.

Es importante tener en cuenta que sólo un proceso puede estar *ejecutándose* en cualquier procesador en un instante dado, pero muchos procesos pueden estar *listos* y *esperando*. El diagrama de estados correspondiente se presenta en la Figura 2.2. Para poder asignar tiempo de CPU a los procesos, debemos especificar primero un orden de ejecución. A este procedimiento se le conoce como planificación. Más adelante en este Capítulo, discutiremos distintas políticas de planificación para manejo concurrente de procesos.

2.4 Tipos de Restricciones de las Tareas de Tiempo Real

Las restricciones que pueden ser aplicadas a las tareas de tiempo real son de tres tipos: restricciones temporales, de relación de precedencia y restricciones de exclusión mutua en recursos compartidos [6].

2.4.1 Restricciones de Tiempo

Los sistemas de tiempo real se caracterizan principalmente por tener restricciones que afectan el tiempo de ejecución de sus actividades. El *plazo de respuesta* (o deadline) es una restricción de tiempo muy común en las tareas de tiempo real, la cual representa el mayor tiempo que una tarea tiene para completar su cómputo sin causar daño al sistema.

Los parámetros asociados a una tarea de tiempo real (τ_i) son:

- **Tiempo de Activación (a_i)**: Es el tiempo en el cual la tarea (τ_i) está lista para su ejecución.
- **Período de Activación (T_i)**: Es el momento en que la tarea (τ_i) realiza una petición de ejecución.
- **Tiempo de Cómputo (C_i)**: Es el tiempo de ejecución de la tarea (τ_i).
- **Tiempo de Inicio (s_i)**: Tiempo en el cual la tarea (τ_i) inicia su ejecución.
- **Tiempo de Finalización (f_i)**: Tiempo en el cual la tarea (τ_i) termina su ejecución.
- **Plazo de Respuesta (D_i)**: Es el tiempo permitido para que la tarea (τ_i) finalize su ajecución.
- **Criticidad**: Es un parámetro relacionado a la consecuencia de la pérdida de plazos de respuesta, o se relaciona con la importancia de las tareas dentro del conjunto de tareas.
- **Latencia (L_i)**: $L_i = D_i - f_i$, representa el retraso en la terminación de una tarea con respecto a su plazo de respuesta. Si $L_i \leq 0$ la tarea (τ_i) no pierde su plazo.

Algunos parámetros temporales definidos se muestran en la Figura 2.3.

Dependiendo de las consecuencias provocadas por las pérdidas de plazos, las tareas de tiempo real se clasifican principalmente en dos clases.

- ***Tareas de tiempo real duras (hard)***. Estas tareas deben cumplir siempre con sus plazos de respuesta, por lo que un fallo en el cumplimiento es intolerable por sus consecuencias en el sistema controlado.

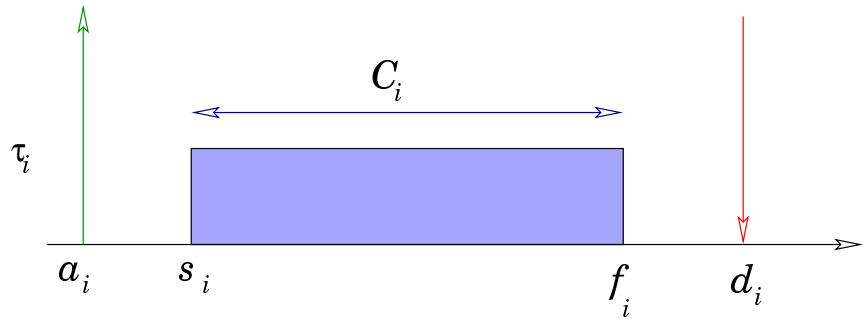

Figura 2.3: Parámetros de una Tarea de Tiempo Real

- ***Tareas de tiempo real suaves (soft)***. Para estas tareas el no cumplir con sus plazos de respuesta, produce una disminución en el rendimiento o calidad de respuesta, pero el funcionamiento se puede considerar todavía correcto.

Otra característica relacionada con los parámetros temporales, concierne a la *regularidad de la activación* de las tareas. Tomando en cuenta la regularidad en la ejecucion de las tareas de tiempo real, estas se pueden clasificar de la siguiente manera:

- **Periódicas**. Las tareas periódicas se ejecutan a intervalos regulares (fijos) llamados instancias. En cada instancia se ejecuta un *Job* de la tarea de tiempo real. Al inicio de cada instancia el job se encuentra listo para ejecución.

- **Aperiódicas**. Las tareas aperiódicas se activan al producirse determinados eventos de forma imprevisible. Las tareas aperiódicas se ejecutan solo durante una instancia de ejecución al termino de la cual desaparecen del sistema. Las tareas aperiódicas, no tienen restricciones críticas.

- **Espóradicas**. Las tareas espóradicas son tareas aperiódicas con restricciones temporales críticas (o duras).

  Si se monitorea el arribo de las tareas esporádicas es posible determinar una separación mínima entre activaciones consecutivas, lo cual podría permitir caracterizarlas como tareas periódicas.

Otras clasificaciones de las tareas de tiempo real son las siguientes:

- **Expulsables y no expulsables**. Las tareas expulsables son aquellas que durante su ejecución pueden ser interrumpidas por el planificador debido a que se necesita ejecutar otra tareas de mayor prioridad. Por otro lado, las tareas no expulsables no podrán ser interrumpidas sino hasta que estas terminen su ejecución.

- **Con restricciones de precedencia**. Las tareas con restricciones de precedencia presentan un orden de ejecución con respecto a otras tareas del sistema. Si la tarea τ_i precede a la tarea τ_j, significa que que la tarea τ_i solo puede ejecutarse hasta que la tarea τ_j termine su ejecución. Este tipo de tareas suelen aparecen en sistema multiprocesadores o sistemas distribuidos, en los cuales varias tareas cooperan para proporcionar la respuesta a los eventos del sistema.

2.4.2 Restricciones de Precedencia

Las aplicaciones de tiempo real, no pueden ser ejecutadas en orden arbitrario ya que tienen que respetar alguna relación de precedencia. Las relaciones de precedencia son descritas por un grafo dirigido acíclico G, donde las tareas son representadas por nodos y las relaciones de precedencia se representan por medio de vértices. Un grafo de precedencia G origina un orden sobre el conjunto de tareas.

- La notación $\tau_a \prec \tau_b$ establece que la tarea τ_a es la antecesora de la tarea τ_b, lo que significa que G tiene un camino directo del nodo τ_a al nodo τ_b.

- La notación $\tau_a \rightarrow \tau_b$ establece que la tarea τ_a es la entecesora inmediata de τ_b, lo que significa que G tiene un vértice directo de la tarea τ_a a la tarea τ_b.

La Figura 2.4 muestra un grafo acíclico que describe las restricciones de precedencia de cinco tareas. Como puede observarse la tarea τ_1 es la única que puede iniciar su ejecución dedido a que no tiene tareas antecesoras. Cuando τ_1 ha completado su cómputo, entonces τ_2 o τ_3 pueden iniciar su ejecución. La tarea τ_4 puede iniciar su ejecución solo cuando τ_2 a terminado, mientras que τ_5 tendrá que esperar hasta que τ_2 y τ_3 terminen su ejecución.

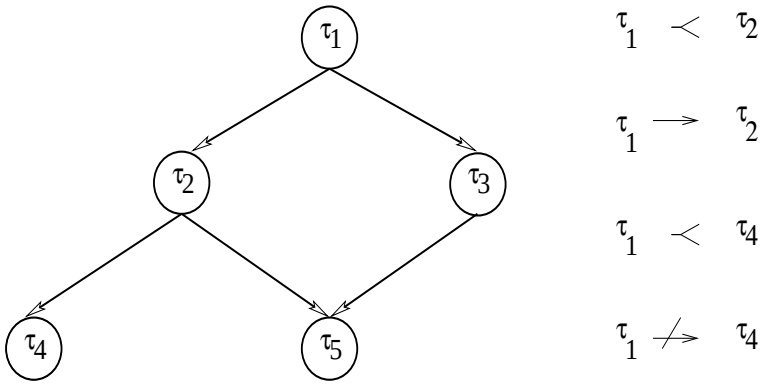

Figura 2.4: Relación de precedencia para cinco tareas.

2.4.3 Restricciones de Recursos

Un recurso es una estructura de software o de hardware que puede ser usada en forma concurrente por varios procesos. Típicamente, un recurso puede ser una estructura de datos, un conjunto de variables, un área de memoria, un archivo, un fragmento de un programa, o un dispositivo periférico. Un recurso dedicado exclusivamente a un proceso particular es llamado *recurso privado ó exclusivo*, mientras que un recurso que puede ser utilizado por una o más tareas es llamado *recurso compartido*.

Cuando varios procesos accesan a un *recurso exclusivo*, su acceso debe de darse en forma ordenada y sincronizada. Esto se debe a que sólo un proceso a la vez puede estar haciendo acceso de esta recurso. Esta situación puede motivar pérdidas de plazos, si algún proceso de baja prioridad hace uso de un recurso exclusivo por largos períodos de tiempo. Cuando se hace uso de *recursos compartidos*, varios procesos a la vez pueden accesar a dichos recursos. En el caso de los recursos compartidos sean datos (o bases de datos), un factor importante a mantener es la consistencia y la integridad en los datos.

Para mantener la consistencia de datos en *recursos compartidos*, no debe permitirse el acceso simultáneo por dos o más tareas a estos datos. En este caso deben implementarse mecanismos de *exclusión mutua* entre las tareas que compiten por este recurso. Supongamos que R es un recurso compartido exclusivo para las tareas τ_a y τ_b. Si A es una operación realizada por τ_a sobre R, y B es la operación realizada por τ_b sobre R, entonces A y B nunca pueden ejecutarse al mismo tiempo. Un fragmento de código ejecutado bajo restricciones de

exclusión mutua se le conoce como *sección crítica*.

Para asegurar el acceso secuencial sobre recursos compartidos, los sistemas operativos normalmente proveen mecanismos de sincronización (tales como semáforos) que puedan ser usados por las tareas para crear regiones críticas.

2.5 Definición del Problema de Planificación

Para definir el problema de planificación necesitamos especificar tres conjuntos: un conjunto de n tareas $\Sigma = \{\tau_1, \tau_2, ...\tau_n\}$, un conjunto de m procesadores $P=\{P_1, P_2, ...P_m\}$ y un conjunto de recursos $R=\{R_1, R_2, ...R_s\}$. Además, es necesario especificar las relaciones de precedencia entre tareas y las restricciones de tiempos de cada tarea. En este contexto, la planificación permite asignar los procesadores del conjunto P y los recursos del conjunto R a tareas del conjunto Σ de acuerdo a un orden que permita completar las tareas bajo las restricciones impuestas. Se ha demostrado que este problema es de tipo NP-completo, lo que significa que la solución es muy difícil de obtener [15].

Para reducir la complejidad en la construcción de planificadores, podemos simplificar la arquitectura de la computadora, por ejemplo, restringiendo al sistema para que utilice un solo procesador, adoptar un modelo con desalojo, usar prioridades fijas, eliminar precedencias o restricciones de recursos, asumir una activación simultánea de tareas y suponer que en el sistema existen conjuntos de tareas homogéneos (únicamente tareas periódicas o tareas aperiódicas).

2.6 Paradigmas de Planificación

Una de las estrategias más conocidas para la planificación de tareas periódicas en un ambiente uniprocesador es el planificador *cíclico* o también conocido como *ejecutivo cíclico*[18]. Esta estrategia de planificación consiste en construir un plan de ejecución que se repite cíclicamente. Este plan de ejecución se divide en un *cíclo principal* T_M, el cual a su vez se divide en *cíclos secundarios* con periodo T_S. En cada cíclo secundario se ejecutan las actividades correspondientes a cada tarea. El principal problema que presentan los ejecutivos cíclicos es la poca flexibilidad en el momento de modificar parámetros (añadir o borrar

tareas), pues ello conlleva a la re-elaboración de todo el plan.

Otro tipo de planificadores, son los basados en *prioridades*[1]. La asignación de la prioridad de una tarea puede realizarse de forma estática (*la asignación de prioridades se realiza al principio de la ejecución del sistema, y no cambia durante la ejecución.*) ó dinámica (*la asignación de prioridades se realiza en forma diámica durante la ejecución del sistema*).

2.6.1 Planificación basada en Prioridades Estáticas

En esta planificación, las prioridades de las tareas se asignan en forma estática, antes de la ejecución del sistema, y no cambian durante su ejecución. En la planificación basada en prioridades estáticas (*off-line*), se realiza un análisis de planificabilidad de las tareas *fuera de linea*. Esta planificación tiene como ventajas:

- Producir sobre-cargas muy bajas, ya que la asignacion de prioridades se realiza una sola vez antes de la ejecución. Las prioridades asignadas a las tareas son guardadas en una tabla la cual permite al planificador decidir el orden de ejecucion de las tareas.
- Permite verificar el comportamiento temporal de las tareas a priori (predecibilidad). Es adecuada para trabajar con tareas con plazos críticos, ya el análisis de planificabilidad nos permite comprobar a priori si dichas tareas cumplirán sus plazos de respuesta.
- En situaciones de sobrecarga del sistema, siempre es posible predecir que las tareas de menor prioridad serán las primeras en perder sus plazos.

Sus principales desventajas son:

- Requiere un conocimiento previo de todos los parámetros de las tareas.
- Es incapaz de tratar adecuadamente las tareas aperiódicas.
- Inflexible a operar bajo ambientes dinámicos en donde los parámetros cambian constantemente.

[1]La prioridad de cada tarea indica el orden de ejecución relativo de la tarea dentro del conjunto.

2.6.2 Planificación basada en Prioridades Dinámicas

En este tipo de planificación las prioridades de las tareas son asignadas durante la ejecución del sistema. Sin embargo, todas las tareas y sus parámetros temporales son conocidos desde el inicio de la ejecución. En la planificación basada en prioridades dinámicas (*on-line*, en línea), se realiza un análisis de planificabilidad fuera de línea.

Sus principales ventajas son:

- Es posible aprovechar mejor el procesador. Debido a que el planificador asigna en forma dinámica las prioridades de las tareas, el CPU puede utilizarse de forma más eficiente que en el caso de la planificación por prioridades estáticas.

y sus principales desventajas son:

- En una sobrecarga del sistema no es posible predecir que tareas pierden sus plazos de respuesta. Lo que es más grave, es que en una sobrecarga, la pérdida de plazos de algunas tareas puede producir un efecto en cascada de perdida de plazos de otras tareas.

- Se incrementa la sobrecarga causada por la planificación. Esto se debe a que la planificación continuamente tiene que ordenar las tareas por orden de prioridad, lo cual introduce sobrecarga al sistema.

En la Figura 2.5 se muestra un diagrama de bloques donde se presenta de manera general la clasificación de los algoritmos de planificación para sistemas de tiempo real bajo un ambiente uniprocesador.

De la gran variedad de algoritmos de planificación de tareas en tiempo real propuestos, podemos identificar las siguientes clases de planificación:

- **Planificación con desalojo**. En esta planificación, las tareas del sistema pueden ser interrumpidas en su ejecución cuando una tarea de mayor prioridad requiere ser ejecutada.

- **Planificación sin desalojo**. En este tipo de planificación solo se permite interrumpir la ejecución de las tareas hasta que éstas terminen su ejecución.

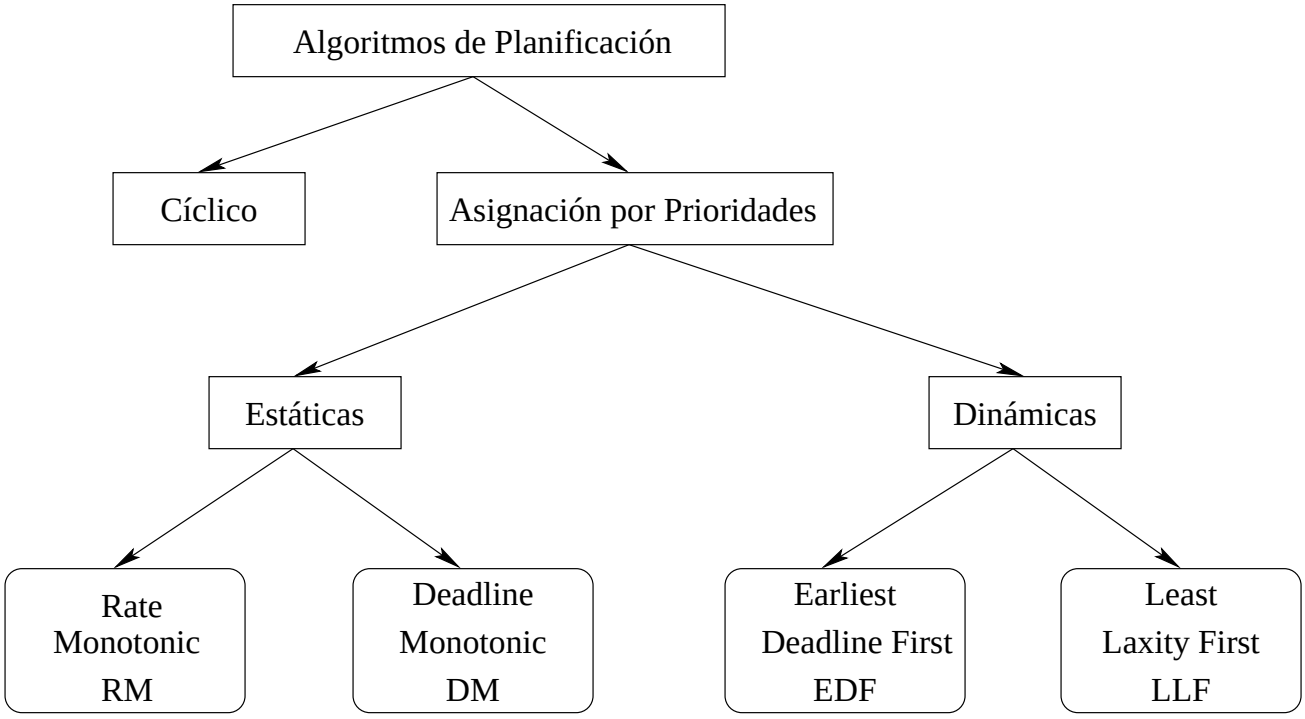

Figura 2.5: Clasificación de los algoritmos de planificación para sistemas de tiempo real.

- **Planificación estática**. Los algoritmos de planificación estáticos son aquellos en los cuales las decisiones de planificación están basados en parámetros fijos, los cuales son asignados a las tareas antes de su activación.

- **Planificación dinámica**. Los algoritmos de planificación dinámicos son aquellos en los cuales las decisiones de planificación están basadas en parámetros dinámicos que pueden cambiar durante la ejecución del sistema.

- **Planificación fuera de línea *(off line)***. Se dice que un algoritmo de planificación es fuera de línea cuando la asignación de prioridades de las tareas se realiza antes de la ejecucución del sistema. La planificación generada en esta forma, es almacenada en una tabla y después ejecutada por un despachador.

- **Planificación en línea (*on-line)***. Se dice que un algoritmo de planificación es en línea si las decisiones de planificación son tomadas en tiempo de ejecución. En este caso, las decisiones de planificación se llevan a cabo cada vez que: (a) Una taera cambia de

estado, (b) Cambian los parámetros temporales del sistema, (c) Se asignan prioridades a las tareas, (d) Una nueva tarea llega al sistema o cuando (e) Una tarea termina su ejecución y sale del sistema.

Se dice que una planificación es *óptima* si minimiza algunas funciones de costo definidas sobre el conjunto de tareas. Cuando ninguna función de costo es definida y lo único concerniente es alcanzar una planificación factible, entonces se dice que un algoritmo de planificación es óptimo si este encuentra una solución de planificación que no puede ser contradecida por ningún otro algoritmo de planificación. La solución de planificación indica si un conjunto de tareas es factible o no factible. Es decir, si cumple o no con sus plazos de respuesta. Por otro lado, un algoritmo de *planificación heurístico* encuentra una solución aproximada que intenta estar cerca de la solución óptima.

Algoritmos con Garantías

En aplicaciones de tiempo real críticas se requiere un comportamiento altamente predecible. El cumplimiento de plazos de respuesta de las tareas del sistema debe ser garantizado en forma anticipada, es decir, antes de la ejecución de las tareas. De esta forma, si una tarea crítica no puede ser planificada dentro de su plazo de respuesta, el diseñador del sistema puede cambiar los parámetros temporales de las tareas u óptimizar el software a fin de lograr una planificación factible. Para comprobar la factibilidad de la planificación antes de la ejecución de las tareas, el sistema tiene que planear sus acciones asumiendo escenarios en el peor caso. Asumir que una tareas se ejecuta con suposiciones de peor caso, implica que siempre considerará sus máximos tiempos de ejecución posible. Por ejemplo, durante distintas instancias de ejecución una tarea puede ejecutarse con distintos tiempos de ejecución, con lo cual el máximo de estos tiempos de ejecución, se consideraría como el peor caso.

En los sistemas de tiempo real estáticos, donde el conjunto de tareas es fijo y conocido, todas las activaciones pueden ser precalculadas fuera de línea, y la planificación completa puede ser almacenada en una tabla con todas las tareas ordenadas y garantizadas. En este caso, durante la ejecución, el despachador simplemente sigue el orden de ejecución de las tareas definido en la tabla. La ventaja principal del modo estático es que el tiempo de ejecución del algoritmo es muy bajo. Esto permite la utilización de algoritmos muy complejos para encontrar secuencias de planificación óptimas. Por otro lado, una desventaja de este

tipo de planificación es que produce un sistema inflexible a cambios en el ambiente, y en donde no se permite que las tareas cambien sus parámetros durante su ejecución..

En sistemas de tiempo real dinámicos, donde nuevas tareas pueden activarse en tiempo de ejecución, la garantía de planificabilidad (*pruebas de aceptación*) debe realizarse en línea cada vez que una tarea entra o sale del sistema.

Debido a que los mecanismos de garantía están basados sobre suposiciones en el peor caso, muchas tareas podrían no ejecutarse. Lo que implica que la garantía de tareas duras se alcanza con un costo de reducción del desempeño promedio del sistema. Por otro lado, el beneficio de tener un mecanismo de garantía es que las situaciones de sobrecarga pueden ser detectadas anticipadamente y evitar efectos negativos en el sistema.

En resumen, una prueba de aceptación debe asegurar que una vez que una tarea es aceptada para ejecución en el sistema, ésta completará su ejecución dentro de su plazo de respuesta y también su ejecución no arriesgará la factibilidad de las tareas que previamente habían sido garantizadas.

Algoritmo de Planificación del Mejor Esfuerzo.

En ciertas aplicaciones de tiempo real, las tareas tienen *restricciones de tiempo suaves*. Sin embargo, en este caso ningún evento catastrófico ocurrirá si una o más tareas pierden sus plazos de respuesta. La única consecuencia asociada con la pérdida de plazos, es la degradación en el desempeño del sistema. Los algoritmos de planificación de mejor esfuerzo intentan siempre cumplir con los plazos de respuesta, sin embargo no proveen garantía de planificación. Estos algoritmos obtienen un mejor desempeño que los esquemas con garantías debido a que no se ejecutan bajo suposiciones pesimistas (de peor caso) en la planificación.

Algoritmos de de Planificación de Cómputo Impreciso.

En un sistema de tiempo real que utiliza algoritmos de planificación de cómputo impreciso, cuaquier tarea τ_i del sistema es dividida en una *subtarea obligatoria* M_i, y una *subtarea opcional* O_i. La subtarea obligatoria es la parte del cómputo que debe ser ejecutado para producir un resultado de calidad aceptable, mientras que la tarea opcional permite refinar el resultado. La subtarea opcional se ejecuta despues de que la obligatoria termina, sin embargo ambas cuentan con el mismo plazo de respuesta d_i. Las subtareas M_i y O_i tienen tiempo de cómputo m_i y o_i, tal que $m_i + o_i = C_i$. Para garantizar un mínimo nivel de

funcionamiento, M_i debe ser completada dentro de su plazo de respuesta (se debe garantizar la planificabilidad de M_i), mientras que parte del cómputo de O_i puede ser desechado, si fuera necesario, a cambio de degradar la calidad del resultado producido por la tarea.

En sistemas que ejecutan algoritmos imprecisos, el error ϵ_i producido por τ_i es definido como la longitud de la parte de la subtarea O_i desechada en la planificación. Si σ_i es el tiempo de procesador asignado a O_i por el planificador, el error de la tarea τ_i es igual a:

$$\epsilon_i = o_i - \sigma_i \tag{2.1}$$

El error promedio $\bar{\epsilon}$ de un conjunto de tareas de define como:

$$\bar{\epsilon} = \sum_{i=1}^{n} w_i \epsilon_i \tag{2.2}$$

donde w_i es la importancia relativa de τ_i en el conjunto de tareas. Un error $\epsilon > 0$ significa que una parte de la tarea O_i ha sido descartada por el planificador arriesgando la calidad producida por la tarea τ_i, pero en beneficio de otras tareas obligatorias para que puedan completar su plazo de respuesta.

En este modelo, una planificación es factible si cada tarea obligatoria es completada en el intervalo $[a_i, d_i]$. La planificación es precisa si el promedio de error $\bar{\epsilon}$ en el conjunto de tareas es cero. En una planificación precisa, el total del cómputo de todas las tareas obligatorias y opcionales son completadas en el intervalo $[a_i, d_i]$.

2.7 Métricas para el Evaluación de Funcionamiento

El funcionamiento de los algoritmos de planificación es típicamente evaluado por una función de costo definida sobre un conjunto de tareas. Por ejemplo: Para los algoritmos de planificación clásicos se trata de reducir al mínimo el tiempo de respuesta medio, el tiempo de finalización total, la suma ponderada de veces de finalización, o el retrazo máximo.

El número de plazos de respuesta pérdidos se considera como una métrica importante en el funcionamiento de un sistema de tiempo real. La tabla 2.1 muestra algunas funciones comunmente usadas para evaluar el funcionamiento de un algoritmo de planificación.

Métrica	Función
Tiempo de respuesta promedio	$\bar{t_r} = \frac{1}{n}\sum_{n}^{i=1}(f_i - a_i)$
Tiempo total de finalización	$t_c = max_i(f_i) - min_i(a_i)$
Suma ponderada de tiempo de finalización	$t_w = \sum_{n}^{i=1} w_i f_i$
Número máximo de retardos	$L_{max} = max_i(f_i - d_i)$
Número máximo de retardos de las tareas	$N_{late} = \sum_{n}^{i=1} miss(f_i)$ donde: $miss(f_i) = \begin{cases} 0 & \text{si } f_i \leq d_i \\ 1 & \text{otro caso} \end{cases}$

Tabla 2.1: Funciones de costo para evaluar los algoritmos de planificación

2.8 Planificación de Tareas Periódicas

En la mayoría de las aplicaciones de sistemas de tiempo real, las actividades periódicas representan la mayor demanda computacional del sistema. Las tareas periódicas típicamente realizan acciones como la adquisición de datos, lazos de control, y monitoreo del sistema. Tales actividades necesitan ejecutarse en forma cíclica a frecuencias fijas y estas frecuencias pueden obtenerse de los requerimientos del sistema. En una aplicación de tiempo real que consiste de varias tareas periódicas concurrentes con restricciones de tiempo, es necesario que el sistema garantice que cada instancia se active regularmente a su propia frecuencia y el cómputo que se realize en cada instancia (Job) finalice antes de su plazo de respuesta.

Para facilitar la descripción de los métodos de planificación, tomaremos como base el siguiente modelo:

τ_i Denota una tarea periódica i.

τ_i^x Denota la instancia x de la tarea τ_i

ϕ_i Denota la fase de la tarea τ_i, el tiempo de activación de la primera instancia.

D_i Denota el plazo de respuesta

T_i Denota el período

C_i Denota Tiempo de Cómputo

d_i^x Denota el plazo de respuesta absoluto de la instancia x de la tarea

$\tau_i(d_i^x = \phi_i + (x-1)T_i + D_i)$

s_i Denota el tiempo de inicio de la instancia x de la tareas τ_i

f_i^x Denota el tiempo de finalización de la instacia x de la tarea τ_i

Γ Denota un conjunto de tareas periódicas.

Un conjunto de tareas Γ tendrá como elementos $\{\tau_1, \tau_2, ..., \tau_n\}$ donde n es la cardinalidad de Γ.

Para simplificar el análisis de planificabilidad, se asumen las siguientes hipotesis:

1. Las instancias de una tareas periódica τ_i son activadas regularmente a una frecuencia constante. El intervalo T_i entre dos activaciones consecutivas es llamado período de la tarea.

2. Todas las instancias de una tarea periódica τ_i tienen el mismo plazo de respuesta D_i, el cual es igual al período T_i.

3. Todas las instancias de una tarea periódica τ_i tienen el mismo tiempo de cómputo (C_i) en el peor caso.

4. Todas las tareas de Γ son independientes, es decir, no tienen relaciones de precedencia ni restricciones de recursos.

5. Ninguna tarea puede suspenderse a sí misma.

6. El tiempo consumido por las operaciones del kernel son despreciables.

El modelo de tareas presentado, aunque es simple, nos permitirá ejemplificar el análisis de los algoritmos de planificación.

2.9 Factor de Utilización

Dado el modelo de tareas presentado en la sección anterior, definiremos el factor de utilización de la siguiente manera:

Definición 2.9.1 *El Factor de Utilización (**U**) indica la fracción o porcentaje de tiempo que el procesador emplea para la ejecución de una tarea, y por lo tanto representa una medida de carga del procesador.*

El factor de utilización para una tarea τ_i se expresa de la siguiente manera:

$$U_i = \frac{C_i}{T_i} \tag{2.3}$$

De la expresión 2.3 se tiene que el factor de utilización para un conjunto de tareas es:

$$U_{TOT} = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.4}$$

De la definición 2.9.1 podemos concluir que para cualquier algoritmo de planificación para sistemas de tiempo real en ambientes uniprocesadores se tiene que:

Teorema 2.9.1 *[16] La condición necesaria aunque no suficiente, para que un algoritmo de planificación sea factible es que:*

$$U_{TOT} \leq 1 \tag{2.5}$$

2.10 Algoritmo de Planificación Cíclico

En la planificación ciclica, todas las tareas del sistema se planifican dentro de un plan de ejecución. En este plan de ejecucion existe un ciclo principal, y varios ciclos secundarios.

La forma de calcular los ciclos principal y secundarios se describe mediante el siguiente ejemplo.

En la Figura 2.6 se presenta la ejecución cíclica para el conjunto de tareas presentado en la tabla 2.2.

Para construir el plan de ejecución, se debe calcular la duración del cíclo principal T_M y la duración de los cíclos secundarios T_S. La duración del cíclo principal corresponde al mínimo común múltiplo de los períodos de las tareas, $T_M = mcm(\tau_i)$. Resultando para el ejemplo de la tabla 2.2, $T_M = 100$. Por otro lado, para calcular el tamaño de los cíclos secundarios, m, se deben considerar las siguientes condiciones:

Tarea	*T*	*C*
A	25	10
B	25	8
C	50	5
D	50	4
E	100	2

Tabla 2.2: Conjunto de tareas periódicas

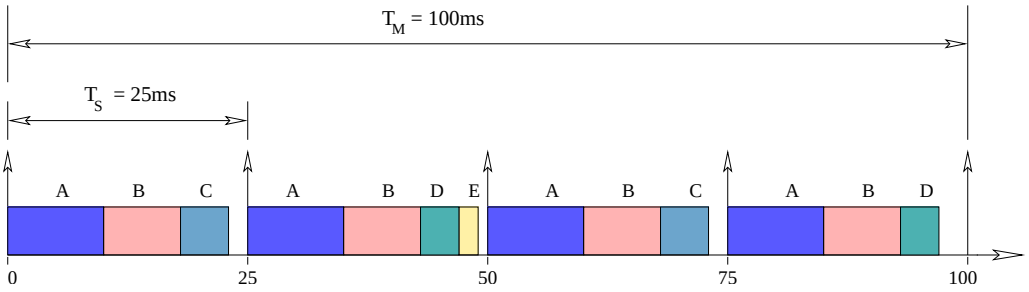

Figura 2.6: Planificación cíclica

- El ciclo secundario debe ser menor o igual que el plazo de ejecución de cada tarea, $m \leq \{D_i\}$
- El ciclo secundario debe ser mayor o igual que el máximo de los tiempos de cómputo de las tareas, $m \geq max\{C_i\}$.
- El ciclo secundario debe ser divisor del período de cada tarea. Es decir, m divide a T_i
- Se debe cumplir que: $2m - mcd(m, T_i) \leq D_i$, la cual es una condición necesaria y suficiente que indica el instante de activación de las tareas.

Aplicando las condiciones anteriores al ejemplo de la tabla 2.2 se obtiene:

- $m \leq \{D_i\} \Longrightarrow m = \{1, 2, 3, ..., 25\}$
- $m \geq max\{C_i\} \Longrightarrow m = \{10, 11, 12, ..., 25\}$
- m divide a $T_i \Longrightarrow m = \{10, 20, 25\}$

- $2m - mcd(m, T_i) \leq D_i \Longrightarrow m = \{10, 20, 25\}$

Con los valores obtenidos para m se puede calcular el número de cíclos secundarios del cíclo principal mediante $N_{T_S} = T_M/m$. Para $m = 10$ se tiene $N_{T_S} = 10$, para $m = 20$ se tiene $N_{T_S} = 5$ mientras que para $m = 25$ se tiene $N_{T_S} = 4$. Como la complejidad aumenta con el número de cíclos secundarios se elige $m = 25$ con lo que se tiene 4 cíclos secundarios por cíclo principal.

Otro dato que se puede calcular es el número de ejecuciones N_{e_i} de cada tarea τ_i en el cíclo principal, mediante la expresión $N_{e_i} = T_M/T_i$. En el ejemplo de la tabla 2.2 en número de ejecuciones de cada tarea es $N_{e_{A,B}} = 4$ para τ_A y τ_B, $N_{e_{C,D}} = 2$ para τ_C y τ_D y $N_{e_E} = 1$ para τ_E.

2.11 Algoritmos de Planificación Estáticos

2.11.1 Rate Monotonic (RM)

En el algoritmo Rate Monotonic, la asignación de las prioridades se obtiene de acuerdo a los períodos de las tareas. A la tarea con menor período, se le asigna la mayor prioridad. En este algoritmo de planificación, el período es igual al plazo.

Fueron Liu y Layland [16] quienes en 1973 propusieron el algoritmo Rate Monotonic (RM), además en [16] demostraron que:

Teorema 2.11.1 *[16] El algoritmo RM es óptimo dentro de los esquemas de asignación de prioridades estáticas.*

Por óptimo debemos entender que si se tiene una planificación factible para un conjunto dado de tareas mediante un algoritmo de asignación con prioridades estáticas, entonces ese conjunto de tareas también será planificable mediante el algoritmo Rate Monotonic.

En el análisis de este algoritmo[16], los autores proporcionan un *control de admisión* [2] de tareas para RM basado en la utilización del procesador. Este control de admisión, compara

[2]El control de admisión es un mecanismo que permite al planificador decidir sobre la aceptación de las tareas en el sistema. En sistemas de tiempo real con planificación estática, este mecanismo se ejecuta solo una vez, al principio de la ejecución del sistema.

la utilización del conjunto de tareas con una *cota límite* que depende del número de tareas del sistema, es decir, un conjunto de n tareas no perderá su plazo si cumple la siguiente condición:

$$U_{TOT} = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1) \tag{2.6}$$

Teorema 2.11.2 *[16] La condición suficiente para que la planificación de un conjunto de n tareas periódicas sea factible mediante el algoritmo RM, es que su factor de utilización total* (U_{TOT}) *cumpla la siguiente condición:*

$$U_{TOT} \leq U_{min} = n(2^{1/n} - 1) \tag{2.7}$$

La tabla 2.3 describe el comportamiento de U_{min} para distintos valores de n.

n	U_{min}
1	1
2	0.8284
3	0.7788
4	0.7568
5	0.7334
.	.
.	.
.	.
∞	0.6931

Tabla 2.3: Utilización mínima garantizada para n tareas

Como se puede apreciar en la tabla 2.3, al aumentar el número de tareas la utilización mínima garantizada converge en:

$$U_{min} = \ln 2 \approx 0.6931 \tag{2.8}$$

Ejemplo: Consideremos el conjunto de tareas[3] τ_1 (30,10), τ_2 (40,10) y τ_3 (50,12) mostrado en la Figura 2.7. El factor de utilización de estas tres tareas es:

$$U_{TOT} = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = \frac{10}{30} + \frac{10}{40} + \frac{12}{50} = \frac{494}{600} \approx 0.82 \tag{2.9}$$

Claramente, la utilización total de este conjunto de tareas es mayor que la utilización mínima garantizada para tres tareas establecida por la condición 2.7 ($0.82 > 0.7788$). Siguiendo la ejecución de las tareas, presentada en la Figura 2.7, es posible observar que la tarea τ_3 del conjunto pierde su plazo de respuesta en el tiempo t=50. Esta pérdida de plazo se debe a que en la primera instancia de ejecución solo puede ejecutarse por 10 unidades de tiempo. Siendo que su tiempo de cómputo es de $C_3 = 12$, dos unidades de tiempo quedan sin ser ejecutandas. Note que la tarea τ_3 posee la menor prioridad del conjunto utilizando el esquema de asignación de prioridades del algoritmo RM.

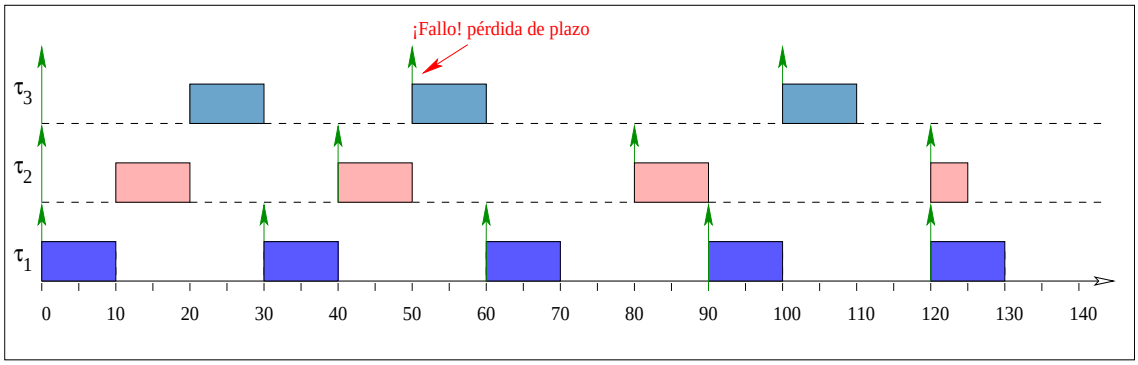

Figura 2.7: Conjunto de tareas que no satisface la Ecuación 2.7 y por lo tanto no es planificable

En la Figura 2.7 se representa el conjunto de tareas mediante una gráfica de Gantt. Las flechas de la gráfica nos indican los instantes donde las tareas solicitan tiempo de procesador. Los rectangulos sombreados muestran la cantidad de tiempo de cómputo utilizando por la tarea.

Por otro lado, consideremos el conjunto de tareas mostrado en la tabla 2.4. Como se

[3]Generalmente en un modelo simple de tareas, una tarea se puede representar como un par ordenado (T_i,C_i), debido a que el plazo de respuesta es considerado igual al período. Es decir, $D_i = T_i$.

Tarea	T_i	C_i	*Utilizacion*
1	16	4	0.250
2	40	5	0.125
3	80	32	0.400
			0.775

Tabla 2.4: Conjunto de tareas, la utilización total cumple con la condición 2.7

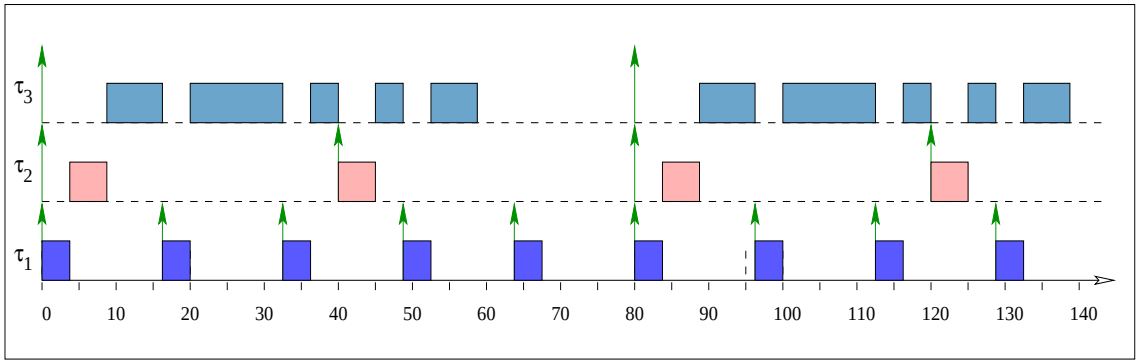

Figura 2.8: Conjunto de tareas que satisface 2.7 y por tanto es planificable

puede observar, la utilización total del conjunto de tareas no excede la cota proporcionada por la condición 2.7, es decir, U_{TOT}=0.775 $<$ 0.778 = U_{min}. Por lo cual se garantiza que este conjunto de tareas no perderá ningún plazo de respuesta. La Figura 2.8 muestra gráficamente el comportamiento del conjunto de tareas.

Tarea	T_i	C_i	*Utilizacion*
1	20	5	0.250
2	40	10	0.250
3	80	40	0.500
			1.000

Tabla 2.5: Conjunto de tareas armónico, es planificable

Es importante observar que la tabla 2.5 describe un conjunto de tareas cuya utilización del procesador es del 100%, es decir, no satisface la condición 2.7, y aún así, el conjunto de tareas cumple con los plazos de respuesta. Es por esta razón, que a esta condición se le

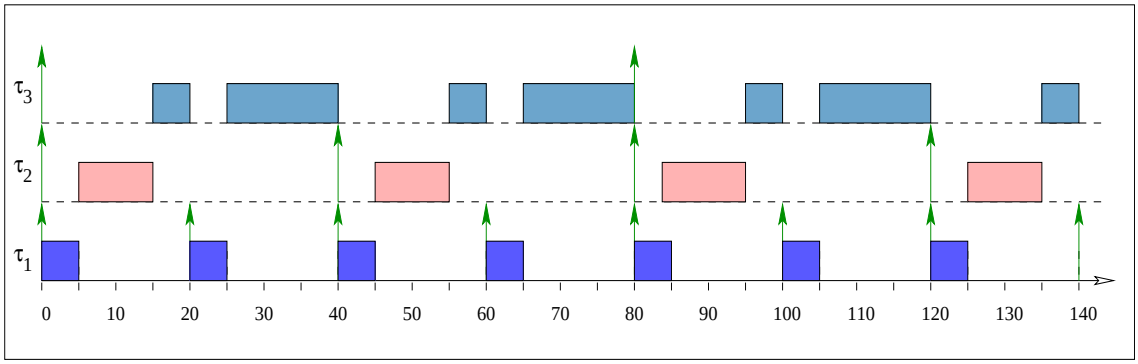

Figura 2.9: Conjunto de tareas que no satisface 2.7. Sin embargo, es planificable

conoce como una condición suficiente, pero no necesaria. La Figura 2.9 muestra la ejecución de las tareas sin que éstas hayan pérdido su plazo de respuesta.

2.11.2 Condición de Planificabilidad Suficiente y Necesaria

Lehoczky et al. [12] desarrollaron una condición de planificabilidad necesaria y suficiente para un conjunto de tareas que es ejecutado en un sistema uniprocesador. Esta condición se define en el teorema 2.11.3

Teorema 2.11.3 *[12] Sea* $\Gamma = \{\tau_1, \tau_2,...,\tau_n\}$ *un conjunto de n tareas periódicas, con* $T_1 \leq T_2 \leq T_3 \leq ... \leq T_n$.
τ_i *es planificable usando el algoritmo Rate Monotonic si y sólo si*

$$L_i = min_{\{t \in S_i\}} W_i(t)/t \leq 1. \tag{2.10}$$

El conjunto entero de tareas es planificable bajo el algoritmo Rate Monotonic si y sólo si

$$L = max_{\{1 \leq i \leq n\}} L_i \leq 1. \tag{2.11}$$

donde:

Los elementos de S_i son los puntos en que la tarea i es planificada. Es decir, los instantes de tiempo en que se cumple el plazo de respuesta y los tiempos de arribo de las tareas de mayor prioridad que i.

$$S_i = \{k \cdot T_j | j = 1, ..., i; k = 1, ..., \lfloor T_i/T_j \rfloor\}. \tag{2.12}$$

$W_i(t)$ mide la cantidad de tiempo de procesador requerido por i tareas, en el instante t.

$$W_i(t) = \sum_{j=1}^{i} C_j \cdot \lceil t/T_j \rceil. \tag{2.13}$$

$$L_i(t) = W_i(t)/t. \tag{2.14}$$

Esta condición de planificabilidad se realiza a partir del instante crítico. El instante crítico es el instante de tiempo en el cual todas las tareas presentan su máximo tiempo de respuesta. Este instante se presenta cuando todas las tareas comienzan a ejecutarse al mismo tiempo.

La condición del teorema 2.11.3 es exacta. Si el conjunto de tareas no es planificable bajo esta condición, se debe a que éste no es planificable (bajo ningún algoritmo de planificación), por lo que en este caso, al menos una tarea perderá su plazo de respuesta.

$Tarea$	T_i	C_i	$Utilizacion$
1	7	3	0.428
2	12	3	0.250
3	20	5	0.250
			0.928

Tabla 2.6: Conjunto de Tareas que no satisface 2.7

Ejemplo: La tabla 2.6 muestra un conjunto de tareas periódicas el cual será planificado con el algoritmo de planificación RM, claramente podemos observar que el factor de utilización total es de 0.928, que es mayor que $U_{min}(3)$. Por tanto, la condición 2.7 no nos permite asegurar nada sobre si se pueden garantizar los plazos de las tres tareas. Sin embargo, el factor de utilización de las dos primeras tareas (τ_1 y τ_2) es igual a $0.678 < U_{min}(2) = 0.828$, por lo que podemos asegurar que τ_1 y τ_2 terminan siempre dentro de sus plazos de respuesta. Se aplica la condición 2.12 para comprobar si el plazo de respuesta de τ_3 está garantizado.

Para ello, se examina la ejecución del sistema en el intervalo de tiempo [0,20], suponiendo que $t = 0$ es un instante crítico. Obteniendose los siguientes puntos de planificación de τ_3:

$$S_3 = \{1 * 7, 2 * 7, 1 * 12, 1 * 20\} = \{7, 12, 14, 20\} \tag{2.15}$$

como se observa en la Figura 2.10, coinciden con los instantes límites de τ_1, τ_2 y τ_3 en el intervalo [0,20]. Aplicando la condición 2.10 se obtiene la cantidad de tiempo de procesador requerido por las i tareas, en el instante t, la tabla 2.7 muestra este hecho.

t	$W_3(t)$
7	(3*1 + 3*1 + 5*1) = 11
12	(3*2 + 3*1 + 5*1) = 14
14	(3*2 + 3*2 + 5*1) = 17
20	(3*3 + 3*2 + 5*1) = 20

Tabla 2.7: Carga en el instante t

Puesto que, al menos en un caso ($t = 20$), se verifica la condición 2.10, podemos concluir que τ_3 está garantizada y por tanto, el conjunto de tareas es planificable.

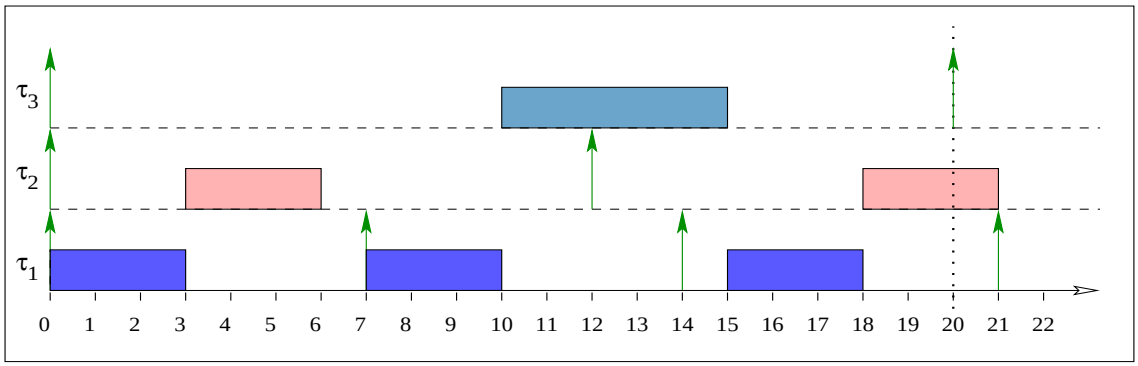

Figura 2.10: Puntos de Planificación

2.11.3 Deadline Monotonic (DM)

En 1982 Leung y Whitehead[14] proponen un esquema de asignación de prioridades fijas denominado Deadline Monotonic, DM, el cual consiste en asignar la prioridad más alta a las tareas que tengan plazos más cortos. Es decir, la tarea con el plazo más corto se le asigna la prioridad más alta.

En DM, el plazo puede ser menor que el período, sin embargo, DM es equivalente a RM cuando el período = plazo de respuesta.

Teorema 2.11.4 *[14] El algoritmo DM es óptimo dentro de los esquemas de asignación de prioridades fijas.*

2.12 Algoritmos de Planificación Dinámicos

2.12.1 Earliest Deadline First (EDF)

Uno de los algoritmos más conocidos para el esquema de asignación dinámica es el algoritmo de planificación guiado por plazos (DDSA: Deadline Driven Scheduling Algorithm), al cual posteriormente se le denominó como el plazo más próximo primero (EDF: Earliest Deadline First).

En el algoritmo EDF las prioridades se asignan en forma dinámica. La política de asignación de prioridades consiste en asignar la prioridad más alta a la tarea con plazo más cercano. Para este algoritmo la condición de planificabilidad se define a continuación.

Teorema 2.12.1 *[16] La condición necesaria y suficiente para que un conjunto de tareas periódicas tenga una planificación factible mediante el algoritmo EDF es:*

$$U \leq 1 \tag{2.16}$$

Esta condición de planificabilidad permite que EDF consiga un factor de utilización del 100% para los conjuntos de tareas que planifica. Por lo que podemos decir que EDF es óptimo globalmente, es decir, que si existe un algoritmo que propocione una planificación factible con un determinado conjunto de tareas periódicas, entonces EDF también proporcionará una planificación factible para dicho conjunto de tareas.

Ejemplo: La tabla 2.8 muestra un conjunto de tareas, cuya utilización total es del 82% y la Figura 2.11 muestra el comportamiento de la ejecución de las tareas bajo el algoritmo de planificación EDF. Como puede observarse no hay pérdida de plazos ya que cumple con la condición 2.16

Tarea	T_i	C_i	*Utilización*
1	30	10	0.333
2	40	10	0.250
3	50	12	0.240
			0.823

Tabla 2.8: Conjunto de tareas, cuya utilización es del 82%

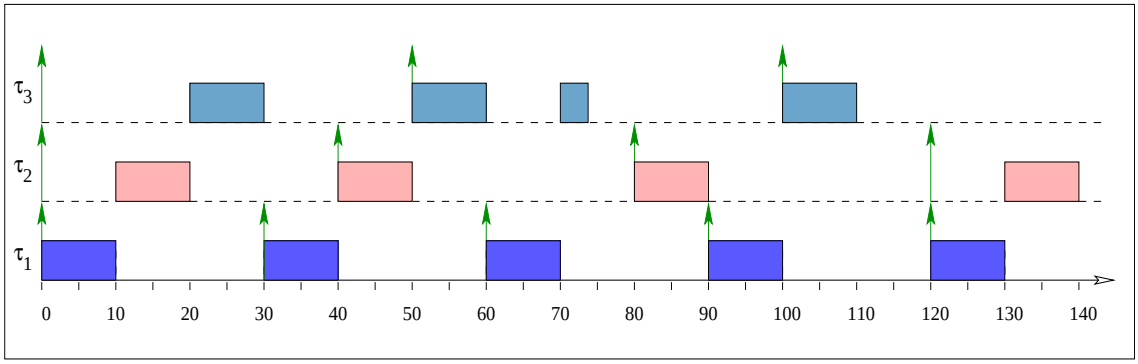

Figura 2.11: Planificación de las tareas de la tabla 2.8 bajo EDF.

2.12.2 Least Laxity First (LLF)

La holgura (*laxity*) de una tarea con tiempo de respuesta D_i en cualquier instante de tiempo t es:

$$holgura = D_i - t - C_i(t) \tag{2.17}$$

donde $C_i(t)$, es el tiempo de cómputo pendiente por ejecutarse para que la tarea termine.

El algoritmo LLF consiste en asignar las prioridades a las tareas de manera inversamente

proporcional a su holgura. Es decir, le asigna la prioridad más alta a la tarea con la menor holgura.

Consideremos el siguiente conjunto de tareas $\Gamma = \{\tau_1$=(6,3), τ_2=(8,2), τ_3=(70,2)$\}$. Para τ_1, en algún instante de tiempo t antes que su tiempo de cómputo finalice, su holgura es: $6 - t - (3 - t)$. Supongamos que la tarea τ_1 es desalojada en el instante de tiempo $t = 2$ por la tarea τ_3 que se ejecuta desde el tiempo 2 al 4. Durante este intervalo, la holgura de τ_1 decrece de 3 a 1. (En el tiempo 4, el resto del tiempo de ejecución de τ_1 es 1, es decir 6-4-1=1). La Figura 2.12 muestra la planificación del conjunto de tareas bajo LLF.

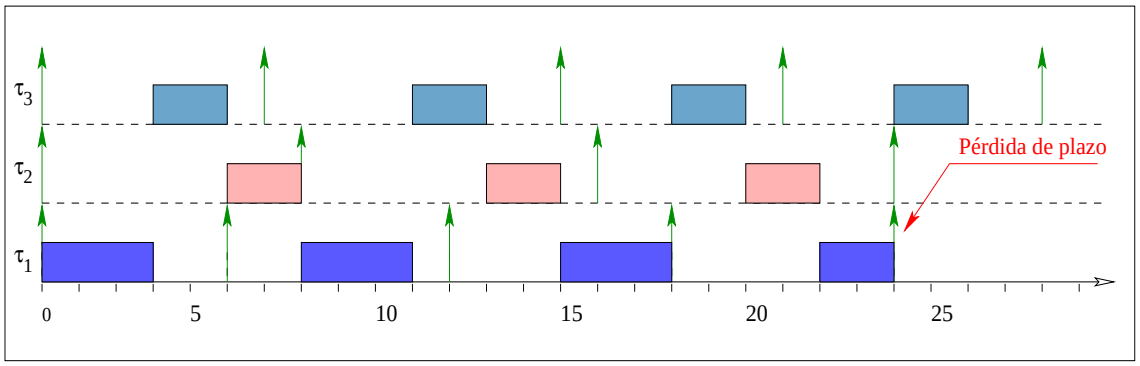

Figura 2.12: Planificación de tareas bajo el algoritmo de planificación LLF.

El algoritmo LLF también puede conseguir, al igual que con EDF, un factor de utilización del 100%.

En el algoritmo EDF no es necesario conocer de los tiempos de cómputo de las tareas, mientras que el algoritmo LLF si se necesita de dicho parámetro temporal. Esto es una deventaja debido a que la holgura es calculada en base al tiempo máximo de cómputo, es decir, la determinación de la holgura es inexacta ya que el algoritmo asume el peor caso para calcular la holgura.

Como se puede observar, los algoritmos con manejo de prioridad dinámica tienen cierta ventaja sobre los algoritmos de prioridad fija. La ventaja radica en el hecho en que la cota límite (máxima) es del 100% para cualquier conjunto de tareas.

Capítulo 3

Trabajos Relacionados

El campo de las herramientas para el diseño, validación y monitoreo de sistemas de tiempo real es muy amplio. Estas herramientas se podrían clasificar de la siguiente forma. Primero están las herramientas de simulación de sistemas de tiempo real, que construyen una planificación en forma gráfica a partir de una especificación detallada de los parámetros temporales de las tareas del sistema. Además estas herramientas pueden simular distintas políticas de planificación estáticas o dinámicas. De la misma forma, existen otras herramientas que además de realizar simulaciones, tiene enlaces con sistemas operativos de tiempo real o con aplicaciones de tiempo real embebidas, que permiten validar las simulaciones hechas. En ambas clasificaciones podemos encontrar herramientas que permiten introducir *scripts* o conjuntos de instrucciones para configurar aplicaciones y para realizar análisis de planificabilidad. También existen herramientas puramente visuales de donde las tareas del sistemas son diseñadas a partir de diagramas de procesos, en donde es posible diseñar las relaciones temporales, de precedencia y de interacción.

En este capítulo presentamos distintas herramientas para diseño y análisis de tareas de tiempo real. Las descripciones fueron tomadas de los artículos donde fueron presentadas.

3.1 ASSERTS

ASSERTS (**A S**oftware **S**imulation **E**nvironment for **R**eal-**T**ime **S**ystems) [11], es un conjunto de herramientas basado en X-Windows para el diseño y desarrollo de sistemas de tiempo real en ambientes uniprocesadores y multiprocesadores. En esta herramienta, los usuarios pueden especificar las tareas, los parámetros del hardware, del kernel del sistema y el tipo de planificador usado en cada procesador. Los parámetros temporales de las tareas del sistema se leen desde archivos proporcionados por el usuario y compilados para tener una representación interna. Estos parámetros pueden editarse y modificarse fácilmente por medio de interfaces gráficas.

Para facilitar el uso de la herramienta, ésta puede leer los parámetros del sistema de cómputo en donde se está ejecutando y hacer el análisis y la simulación apropiada.

ASSERTS es capaz de mostrar las tareas, el procesador (o nodo), y la interconexión entre tareas (los procesadores y los recursos utilizados) por medio de gráficas de Gantt en ventanas independientes. Para lo cual, ASSERTS usa un esquema de colores para mostrar los distintos estados de estas entidades. La visualización de distintas ventanas se puede sincronizar a fin de determinar y verificar la concurrencia de eventos relacionados, en intervalos de tiempo. Adicionalmente, ASSERTS proporciona detalles estadísticos sobre el comportamiento del sistema. En ASSERTS se puede usar también para estudiar el impacto de usar interconexiones, algoritmos de planificación, Kernels, procesadores, estrategias de administración de recursos y esquemas de asignación de tareas alternativos.

3.1.1 Especificación de componentes del Hardware y el Kernel

En ASSERTS es necesario especificar el tipo de procesadores a utilizar y la interconexión entre estos. La especificación para el procesador incluye su velocidad relativa (con respecto al CPU utilizado en el diseño), la capacidad de RAM y los parámetros del kernel (por ejemplo, tiempo de cambio de contexto, los parámetros que caracterizan la sobre-carga de paso de mensajes, el tiempo de inicio de la tarea, el protocolo de acceso a recursos, etc.). También, en ASSERTS es necesario especificar la forma de como se encuentran conectados los CPUs (parámetros físicos tales como ancho de banda, y velocidad de propagación).

Algunas interconexiones simuladas por ASSERTS son las siguientes: (a) Interconexión

Futurebus, el modelo simula protocolos de acceso y transferencia basados en el estándar IEEE [1]. La interconexión Futurebus es muy popular en sistemas multiprocesadores para sistemas de tiempo real, y (b) una interconexión con otros sistemas mediante conexiones utilizando Ethernet, y (c) una interconexión completamente conectadas.

3.1.2 Especificación de algoritmos de planificación

ASSERTS proporciona un conjunto de algoritmos de planificación para su ejecución en cada procesador (nodo). Tiene la capaciad de ejecutar diferentes algoritmos en los distintos nodos que se estén utilizando. Los algoritmos con los que cuenta ASSERTS son: (a) Rate Monotonic (RMS), (b) Rate Monotonic Genérico (GRMS) el cual se usa para ambientes distribuidos[22], (c) Earliest Deadline First (EDF), (d) Ejecutivo cíclico (CE) y (e) Un algoritmo de planificación genérico con manejo de prioridades fijas. Además, de estos algoritmos de planificación implementados se permite agregar nuevos algortimos diseñados por el usuario.

3.1.3 Especificación de las tareas

En ASSERTS es necesario especificar dos componentes para cada tarea del sistema: sus *atributos* y su *representación*.

Los atributos que se utlizan son: El tipo de tarea (periódica, aperiódica), su período (si la tarea es periódica), su plazo de respuesta, la prioridad (asignada por el algoritmo de planificación basado en prioridades fijas), tamaño de la tarea (la cantidad de RAM requerida por la tarea para ejecutarse). Un atributo especial es, *wait type* el cual nos indica si la tarea está en espera, dormida o está bloqueada. Otros dos atributos para las tareas son el tiempo estimado de ejecución y el tiempo estimado de bloqueos. Estos atributos se usan en las pruebas de planificabilidad. Algunos atributos son opcionales [7] y cuentan con valores predefinidos.

La representación de una tarea determina su comportamiento y esta compuesta de varias macro-instrucciones. Cada macro-instrucción describe una fase de la tarea. La Tabla 3.1 presenta un resumen de las macro-instrucciones utilizadas en ASSERTS. Cualquier representación de tarea siempre inicia y finaliza con las macro-instrucciones *start_task* y *end_task*, respectivamente. *start_task* modela el instante de tiempo en que es asignada la tarea al CPU

para ejecutarse, y *end_task* actúa simplemente como un símbolo que delimita el fin de la tarea.

ASSERTS permite que la representación de las tareas puedan ser detalladas en varios niveles de abstracción. El nivel más alto para la representación de una tarea consiste simplemente de *start_task*, una fase de ejecución y *end_task*. La duración de la fase de ejecución es determinada como una cantidad fija (usando $compute < duration >$) ó como una variable aleatoria con límite inferior y superior (usando $rcompute < lower\_bound >, < upper\_bound >$). Las macro-instrucciones permiten al usuario definir la representación de la tarea al nivel necesario que se requiera para el diseño.

Macroinstrucción	Función
start_task	Inicio de la representación de la tarea
end_task	Fin de la representación de la tarea
compute	Tiempo de cómputo que fue asignado a la tarea por el usuario
rcompute	Tiempo de cómputo asignado a una tarea, aleatoriamente
send	Envía mensajes a otras tareas
recive	Recibe mensajes de otras tareas
delay	Especifica el tiempo ocioso
assign	Asigna un valor a la variable compartida
wait	Espera hasta que el valor de una variable compartida cumpla cierta condición
lock	Bloquea una variable
unlock	Libera una variable
set_timer	Establece una variable para controlar el tiempo
wait_timer	Espera hasta que el período haya terminado
if	Realiza una función bajo cierta condición
goto	Realiza un salto sin condición
activate_task	Activa otra tarea para su ejecución
deactive_task	Desaloja una tarea de su ejecución

Tabla 3.1: Macro-instrucciones de ASSERTS.

3.1.4 Activación

Las activaciones son modelos de los eventos externos al sistema. Estos ocurren aleatoriamente o son definidos por el usuario. Cada activación está asociada a una tarea, y cuando ocurre la activación la tarea está lista para ser ejecutada.

ASSERTS cuenta con los siguientes modelos de activación: (a) Activaciones estócasticas: los tiempos entre activaciones sucesivas son pseudo-aleatorias, y son generadas en base ha una función de distribución de probabilidad (Poisson, Uniforme, Normal). (b) Archivo de activaciones: los tiempos de activación se encuentran definidos en un archivo y (c) Activaciones desde la interfaz de usuario: las activaciones son definidas a traves de la interfaz gráfica de usuario.

3.1.5 Interfaz Gráfica de Usuario

La interfaz gráfica de usuario de ASSERTS provee facilidades para cargar, vizualizar, y almacenar las características de la simulación del sistema. Además, realiza una prueba de planificabilidad, y muestra los resultados de la simulación en diferentes gráficas de Gantt.

3.2 PERTS

La herramienta PERTS (**P**rototyping **E**nvironment for **R**eal-**T**ime **S**ystems) [19] contiene una amplia biblioteca de algoritmos de planificación, protocolos para control de acceso a recursos y protocolos de comunicación. Además proporciona un conjunto integrado de herramientas para el análisis y simulación de sistemas de tiempo real.

3.2.1 Modelo de Sistemas de Tiempo Real en PERTS

El propósito de PERTS es validar las restricciones de tiempo y evaluar el comportamiento de los sistemas de tiempo real. Para ello los autores representan un sistema de tiempo real con un grafo de tareas, un grafo de recursos y un conjunto de algoritmos de planificación y protocolos de control de acceso a los recursos, como se muestra en la figura 3.1. Los grafos que representan las tareas de la aplicación son llamados *sistema de tareas*. El grafo de recursos describe los recursos físicos y lógicos disponibles para el sistema de tareas. Los algoritmos de

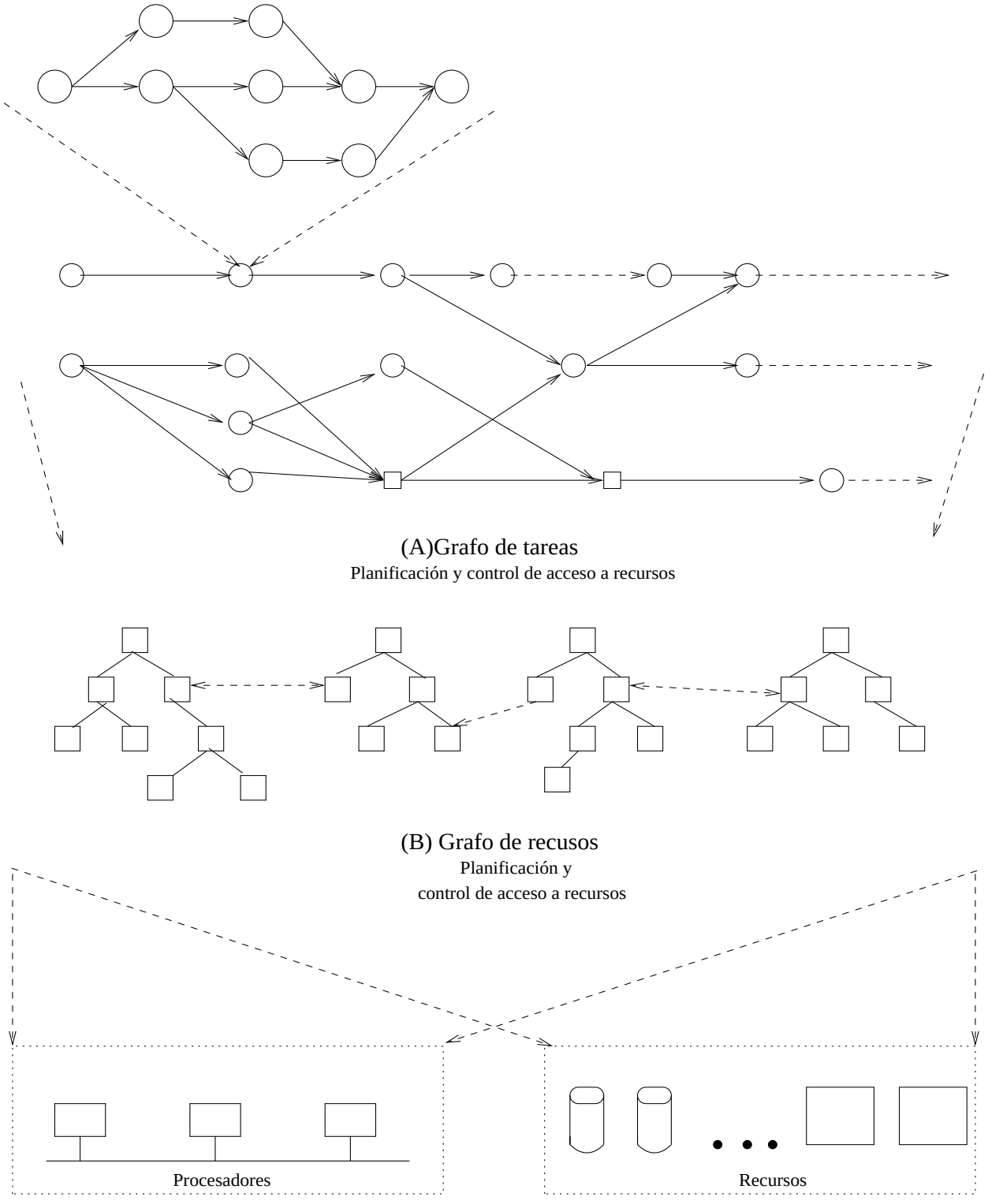

Figura 3.1: Un modelo de referencia para sistemas de tiempo real en PERTS

planificación y algoritmos de control de acceso de recursos caracterizan la parte del sistema operativo que asigna los recursos al sistema de tareas.

3.2.2 Vista general de PERTS

El grafo de tareas es una integración de tres modelos de carga de trabajo usado en aplicaciones de tiempo real: (1) El modelo de tareas periódicas, (2) el modelo de tareas complejas y (3) El modelo de cómputo impreciso [20]. El conjunto de tareas consiste de n tareas y el número de procesadores que se ocuparán para la simulación puede ser uno o más.

El hardware y el sistema es modelado por medio de colas llamadas colas de recursos activos y colas de recursos pasivos. Los procesadores son entidades que son modelados como servidores o recursos activos. Los buses de entrada/salida, comunicación y conexiones virtuales de red son ejemplos de recursos activos. Mientras que dentro de los recursos pasivos se encuentran: Buffers de E/S, semáforos, etc.

Para ejecutar el sistema a simular, se requiere de por lo menos un procesador y un algoritmo de planificación. Además se debe tener en cuenta que cada tarea puede requerir de distintos recursos durante su ejecución, por lo que una tarea τ_i puede entrar en conflicto con otra τ_j si las dos ocupan el mismo recurso y lo necesitan en el mismo instante de tiempo. Para evitar este tipo de conflictos se han implementado los protocolos de acceso a recursos

3.2.3 Grafos de las Tareas y los Recursos

Un grafo de tareas en PERTS es un grafo extendido con precedencia. Cada tarea del sistema de tareas es representado por un nodo en el grafo. Los vértices dirigidos representan datos, restricciones temporales y control de dependencias entre tareas. La figura 3.1 muestra un grafo de tareas. Además, un nodo puede estar compuesto por un subconjunto de tareas como se muestra en la figura 3.1(A).

Cada tarea esta compuesta por cuatro tipo de parámetros:

- **Parámetros temporales**. Éstos se refieren a las restricciones temporales.
- **Parámetros funcionales**. Éstos definen propiedades tales como si la tarea es o no desalojable.

- **Parámetros de recursos**. Éstos indican que recursos requiere la tarea para su ejecución. Cada tipo de recurso requerido por una tarea le corresponden parámetros como en intervalo de tiempo que será usado por la tarea.

- **Parámetros de interconexión**. Los parámetros de interconexión junto con los parámetros de los vértices adyacentes, nos indican si la tarea depende de otras para su ejecución. Por ejemplo, *in_type* establece el tipo de nodo donde se encuentra la tarea, el tipo puede ser AND o OR, para el tipo de nodo AND la tarea estará lista para su ejecución si todos los nodos antecesores han completado la ejecución de sus tareas. Mientras que para el tipo de nodo OR ejecutará la tarea si alguno de los nodos antecesores ha completado la ejecución de su tarea.

La figura 3.1(B) muestra cuatro árboles con nodos cuadrados. Como se puede observar, en la figura hay dos tipos de vértices. Los vértices que nos dan información del recurso se encuentran representados por la líneas sólidas y los vértices con líneas punteadas representan una relación de acceso (nos dicen que recursos se pueden acceder remotamente y por quién).

3.2.4 Jerarquía de Planificación

El tercer elemento en el modelo de PERTS se refiere al conjunto de algoritmos y protocolos que son usados para trazar los grafos de las tareas y los recursos. PERTS cuenta con una variedad de algoritmos y protocolos para la planificación como son los algoritmos de planificación para tareas periódicas y servidores para manejar tareas aperiódicas junto con las tareas periódicas. Además cuenta con algoritmos fuera de línea y en línea para la planificación de tareas de cómputo impreciso. Algunos recursos, tales como la memoria son entidades físicas, mientras que otros recursos como el bloqueo de datos y llamadas al sistema son entidades lógicas. Los recursos lógicos son implementados en el sistema y por tanto deben ser planificados para ejecutarse en recursos físicos.

3.3 STRESS

STRESS es un ambiente de simulación para sistemas de tiempo real duros, que proporciona las herramientas para el diseño y análisis de tareas de tiempo real capaz de soportar las

características del hardware y el kernel del sistema.

El código presentado en la figura 3.2, muestra como planificar un conjunto de tareas de tiempo real utilizando STRESS. La arquitectura del hardware está compuesta de un solo nodo (*node_1*), que contiene un procesador (*proc_1*). El procesador maneja un semáforo binario (*S0*) y dos tareas periódicas (*J0, J1*). Cada tarea tiene un período, un plazo de respuesta y un desplazamiento. La tarea J0 arribará en tiempos 4, 19, 34,... con plazos de respuesta 19, 34, 49,... respectivamente. Cada tarea consume tiempo de procesador, y es bloqueada o liberada por el semáforo. La notación [n,m] indica que la tarea debe consumir entre n y m unidades de tiempo de procesador. Pero cuando n y m son iguales, el tiempo asignado de procesador será n, esto significa que para el caso de la figura 3.2 para [4,4] indica que la tarea consumirá cuatro unidades de tiempo de procesador. Las instrucciones p(S0) y v(S0) representan el bloqueo y la liberacón del semáforo S0.

El esquema de planificación utilizado es la asignación de prioridades fijas, lo cual se establece mediante la instrucción *scheduler pripre*. La política de planificación en este caso es Deadline Monotonic la cual se establece por la intrucción *order_dma*. El protocolo de acceso a recursos utilizado para el ejemplo es *inherit* (por herencia de prioridades).
Una vez escrito el programa de la figura 3.2, se pueden ejecutar tres funciones.

1. Analizar la planificabilidad del conjunto de tareas dado el algoritmo de planificación y el kernel del sistema.

2. Simular el conjunto de tareas tomando en cuenta la arquitectura del hardware.

3. Mostrar los resultados de la simulación.

Para el análisis de planificabilidad se puede seleccionar entre las siguientes pruebas:

1. La prueba de planificabilidad suficiente de Liu y Layland[16].

2. Prueba de planificabilidad suficiente y necesaria de Lehozcky[13].

3. Prueba de planificabilidad de Audsley[4].

4. Prueba suficiente y necesaria de Audsley[4].

```
1. system
2.  node node_1
3.      processor proc_1
4.          order dma
5.          scheduler pripre
6.          resource inherent
7.          semaphore S0
8.              periodic J0
9.                  period 15 deadline 15 offset 4
10.                 [1,1] p(S0) [1,1] v(S0) [1,1]
11.             endper
12.             periodic J1
13.                 period 20 deadline 20 offset 0
14.                 [2,2] p(S0) [4,4] v(S0) [1,1]
15.             endper
16.     endpro
17. ennod
18. endsys
```

Figura 3.2: Programa sencillo utilizando STRESS

3.3.1 Ejecución del simulador

La ejecución del simulador se inicia escogiendo la opción *Simulate* y tiene una duración de 100 ticks, valor que viene predefinido para la simulación. Si estos ticks no son suficientes, el valor puede modificarse utilizando *execution start* y *stop boxes*.

La simulación se inicia seleccionando *simulate* de la ventana del simulador. Los resultados de la simulación son escritos en archivos que serán usados posteriormente por la herramienta gráfica, la figura 3.3 muestra la interfaz gráfica de STRESS.

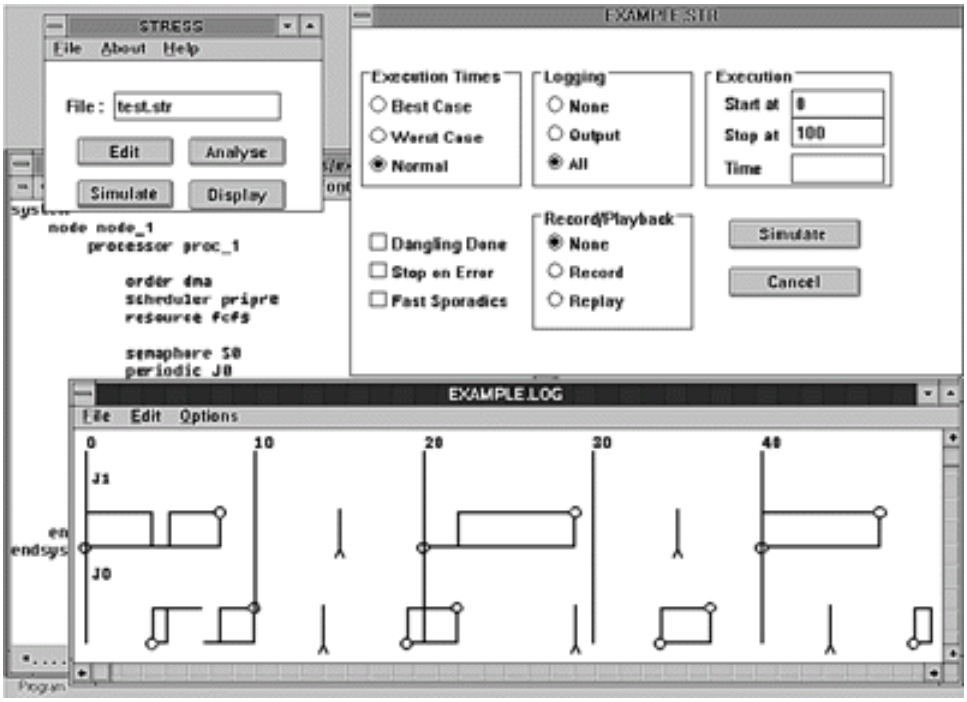

Figura 3.3: Interfaz gráfica de STRESS

3.3.2 Redes, Buzones, Nodos y Procesadores

Los nodos en el sistema son conectados punto a punto. Esto impone pocas limitaciones ya que algunas topologías de red pueden ser simuladas como una restricción sobre la conectividad total. La comunicación sobre la red se representa por medio de mensajes. Un mensaje es enviado por una tarea a un buzón, y una tarea puede recibir mensajes de un buzón. Los buzones tienen un tamaño asociado que se puede declarar explícitamente (el tamaño representa el número de mensajes que pueden ser almacenados en el buzón por algún tiempo), STRESS tiene predefinido el tamaño del buzón con un valor de uno.

3.3.3 Tareas

Una tarea en STRESS es la unidad básica de concurrencia. Las tareas que se pueden implementar son los tres tipo básicos, periódicas, aperiódicas y esporádicas. Las tareas periódicas son declaradas con la instrucción *periodic* y se finaliza con *endper* (de igual manera las tareas aperiódicas se declaran entre las instrucciones *aperiodic* y *endape*). Dependiendo del tipo

de tarea, se pueden declarar otras características. Por ejemplo para las tareas periódicas se requiere del período y el plazo de respuesta.

3.4 GHOST

GHOST (**G**eneral **H**ard real-time **O**riented **S**imulator **T**ool)[21], es una herramienta para la simulación y analísis del comportamiento de un conjunto de tareas de tiempo real mediante un algoritmo de planificación.

La estructura general de la herramienta se presenta en la Figura 3.5, cada caja representa los módulos de la aplicación. La herramienta recibe como entrada un archivo de texto que contiene la descripción del experimento a simular, el archivo es procesado por el *intérprete*, el cual proporciona los datos necesarios al *generador de carga*. Basado en la información proporcionada por el intérprete el generador de carga crea un conjunto de tareas para la simulación y produce las instancias[1] de la tarea a ejecutar. El simulador interactúa con el planificador, el cual planifica las próximas instancias a ser seleccionadas para su ejecución.

El simulador puede producir opcionalmente las secuencias detalladas de una simulación particular en archivos, lo que permite depurar y análizar la planificación. Las secuencias de planificación pueden ser analizadas fuera de línea para determinar alguna falla en la implementación del planificador o para comprender mejor el algoritmo de planificación bajo ciertas condiciones de carga. La información contenida en el archivo de secuencias de planificación permite examinar la ejecución, las colas del sistema, colas de usuario y parámetros de tareas. La herramienta de visualización gráfica proporciona la ejecución para tal análisis. La figura 3.4 muestra la interfaz gráfica de GHOST.

El módulo del generador de salida es llamado cada vez que se tiene un evento particular. Este módulo interactúa con un módulo de especificación de salida definido por el usuario, reúne los datos, extrae los valores estadísticos necesarios y los almacena en un archivo de salida.

[1] Las tareas periódicas consisten de una secuencia de actividades de cómputo, llamadas instancias, que son activadas a una frecuencia fija (período)

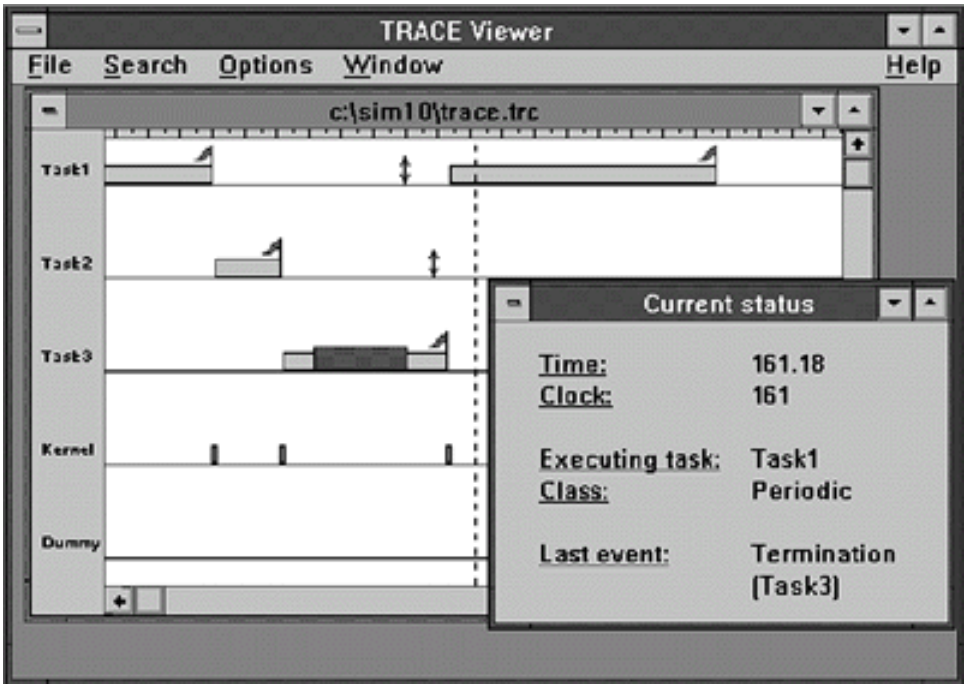

Figura 3.4: Interfaz gráfica de GHOST

3.4.1 Modelo de la Máquina Virtual

GHOST ha sido diseñado para la simulación y análisis de conjuntos de tareas de tiempo real en sistemas uniprocesadores. En el nivel más bajo llamado (*CPU level*), el simulador proveé de un modelo de máquina que es usada para construir un sistema operativo de tiempo real virtual. En este nivel, GHOST realiza la ejecución de un pseudo-código e implementa mecanismos de interrupción basado en prioridades que permite la planificación del tiempo. El pseudo-código ejecutado por el procesador virtual no realiza alguna operación significativa, a parte de consumir tiempo e invocar las primitivas del kernel para acceder a los recursos compartidos. Cada pseudo-instrucción provoca que el simulador evalúe el tiempo asignado contra el tiempo consumido por las tareas.

Para el estudio del desempeño de varios algoritmos de planificación, la noción de tiempo se ha proporcionado en dos resoluciones: *tick mayor* y *tick menor*. El *tick mayor* define la unidad de tiempo para expresar los parámetros temporales de las tareas y es la base para el mecanismo de interrupción y para la rutina de reloj que invoca periódicamente el planificador. El *tick menor* representa la resolución de tiempo y es la cantidad más pequeña que puede ser

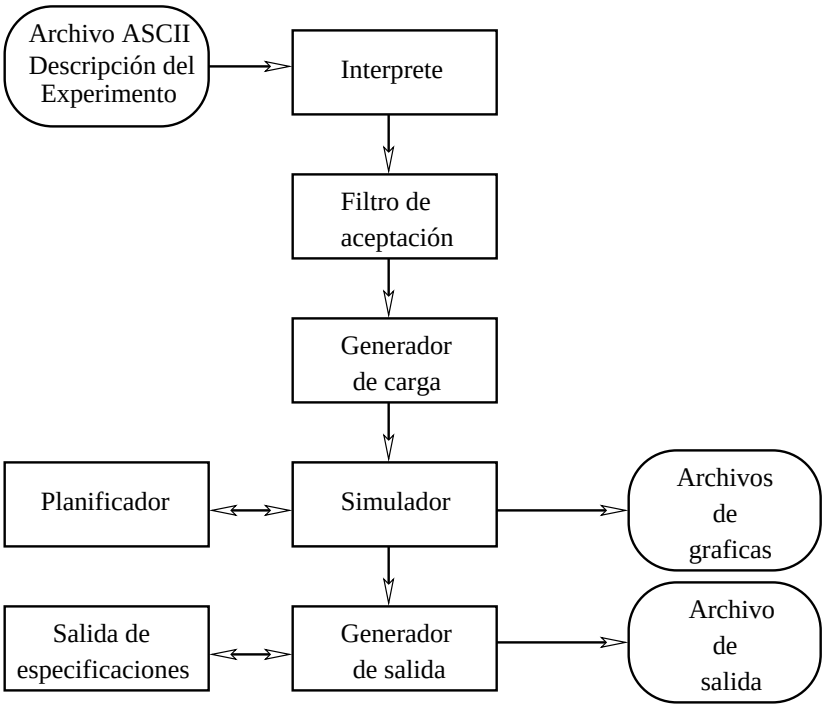

Figura 3.5: Estructura general de la herramienta GHOST

ejecutada por el simulador.

En el siguiente nivel (el nivel del kernel), el simulador proporciona la noción de tarea y las primitivas para la planificación, tales como creación, finalización y cambio de contexto. Se proporcionan las diferentes clases de tareas y un modelo de protocolo para acceder los recursos compartidos. La *clase task* denota un conjunto de tareas con los mismos parámetros y mecanismos de generación. Por ejemplo, el usuario puede definir las tareas periódicas y tareas aperiódicas como dos clases diferentes.

3.4.2 Algoritmos de planificación en GHOST

El algoritmo de planificación es definido por el usuario e insertado por encima de la capa del kernel, Este algoritmo se encuentra estructurado como un conjunto de funciones en C, cada vez que es utilizado es llamado por eventos específicos, tales como el arribo, la finalización de cómputo o la pérdida de plazo de una tarea. El desarrollo de un algoritmo de planificación es respaldado por una extensa biblioteca de GHOST, que proporciona un conjunto de funciones

Función	Descripción
Time()	Devuelve el tiempo actual en ticks secundarios
Clock()	Devuelve el tiempo actual en ticks principales
ExeTask()	Devuelve la tarea que se esta ejecutando
Class(task)	Devuelve la clase de tarea
Period(task)	Devuelve el período de una tarea
Deadline(task)	Devuelve el plazo de respuesta de una tarea
SysQueue(class)	Devuelve la cola asociada con una clase
TaskInsert(task, queue)	Inserta una tarea en la cola
TaskExtract(task, queue)	Extrae un tarea de la cola
FirstTask(queue)	Devuelve la primer tarea de la cola
SetTaskPrio(task, p)	Pone la prioridad a una tarea
TaskSchedule(task, class)	Seleciona una tarea de la clase
ClassSchedule(class)	Seleciona una clase para su ejeción
Lock(res, task)	Bloquea un recurso a una tarea
Unlock(res)	Libera el recurso

Tabla 3.2: Funciones básicas disponibles en GHOST

para analizar el estado del sistema, manejo de colas y para la planificación y manejo de recursos. La tabla 3.2 muestra las funciones de GHOST.

3.4.3 Generador de carga

El archivo que contiene la descripción del experimento es analizado por el intérprete, el cual se encuentra diseñado para describir datos estadísticos sobre el comportamiento del conjunto de tareas. Para describir las características estadísticas del conjunto de tareas, se proporcionan operaciones con arreglos y funciones de distribución estadística. Cuando se define un conjunto de tareas, se crean los arreglos de las tareas los cuales pueden ser asociados con variables que describen los parámetros de las tareas durante la ejecución.

3.4.4 Módulo de salida

Debido a la variedad de planificadores y su desempeño para determinados sistemas, GHOST proporciona un conjunto de estructuras de datos para registrar los valores estadísticos de los parámetros que el usuario necesita controlar durante la simulación.

```
#include "ghost.h"
EVENT_REC resp_time      /* tiempo de respuesta */
TIME_REC queue_len    /* longitud de la cola */
void TestTaskArr() {
        AddTimeRec(&queue_len, 1);
}
void TastTaskEnd(TASK t) {
        UpdEventRec(&resp_time, Time() - Arrival(t));
        AddTimeRec(&queue_len, -1);
}
void TestInit() {
        /* Inicio de las estructuras de registro */
        EventRec(&resp_time, "rtime", MAX_R);
        TimeRec(&queue_len, "qlen", AVG_R);
        Monitor(ARRIVAL, TestTaskArr);
        Monitor(END, TestTaskend);
}
```

Figura 3.6: Ejemplo de un módulo de especificación de salida definido por el usuario en GHOST

La figura 3.6 muestra un ejemplo de un módulo de especificación de salida desarrollado por el usuario indicando lo que se requiere controlar. La macro EVENT_REC es usada para declarar variables que llevan el registro de cuantas veces se ha generado un evento particular (por ejemplo, el tiempo de respuesta y la finalización de las tareas). Por otro lado, TIME_REC es usada para declarar variables que monitorean cantidades en función del tiempo (por ejemplo, la longitud de la cola, o un servidor aperiódico). Una vez que se

ha declarado una variable de registro se le asigna una etiqueta que haga referencia a dicha variable en el archivo de salida. Además, se indica el tipo de parámetro estadístico que se registrará en la variable (mínimo, máximo, promedio o desviación estándar) del evento que se desea controlar.

En el ejemplo de la figura 3.6 la función EventRec(&resp_time, "rtime",MAX_R), asocia la etiqueta **rtime** a la variable **resp_time**, y registra el valor máximo del tiempo de respuesta. La función MONITOR(event, userfun) vincula la función definida por el usuario a un evento. En particular, la función TestTaskArr() incrementa el valor de la variable queue_len cada vez que las tareas arriban. Similarmente, la función TestTaskEnd() monitorea las variables resp_time y queue_len. En general, el ejemplo de la figura 3.6 muestra un módulo de especificación de salida sencillo. Sin embargo, este tipo de módulos puden ser más complejos dependiendo de las necesidades del usuario.

3.5 ARTISST

3.5.1 Características

ARTISST (**A**RTISST is a **R**eal-**T**ime **S**ystem **S**imulation **T**ool)[10], es un ambiente de trabajo modular dedicado a la simulación de sistemas de tiempo real. La figura 3.7 presenta varias entidades que ARTISST es capaz de simular. Una aplicación se ejecuta una capa superior del sistema operativo de tiempo real, e interactúa con un ambiente externo por medio de eventos.

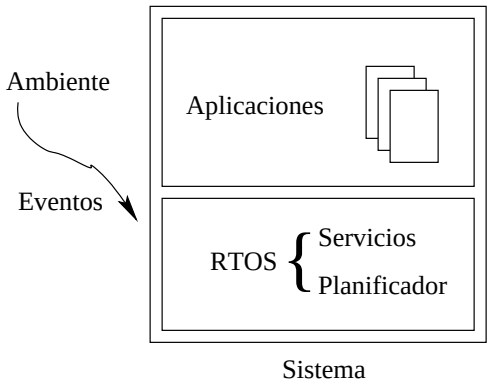

Figura 3.7: Modelo de simulación de ARTISST

ARTISST muestra las características de un sistema que permite al sistema simulado tener un comportamiento cercano al sistema del mundo real. Para permitir que los sistemas complejos sean simulados, se ponen menos restricciones en el lenguaje de programación para los cómputos de las tareas simuladas y los servicios del sistema operativo de tiempo real. Las tareas y el sistema operativo de tiempo real se pueden escribir en C estándar. Además, el simulador cuenta con una primitiva central de simulacion $hold\_cpu(delay)$ que es la responsable de simular el tiempo que se encuentra ocupado el CPU. Esta primitiva es importante debido a que puede ser empotrada en el código de alguna aplicación.

3.5.2 Modularidad

La figura 3.8, muestra que el sistema simulador constituye una parte del ambiente de trabajo. Es decir, el simulador de ARTISST se ejecuta con una interconexión de módulos definida por el usuario. La interconexión de módulos se hace por medio de intercambio de mensajes. Estos mensajes reflejan la ocurrencia de interrupciones de hardware (simuladas), el principio y finalización de estas interrupciones y la creación/desalojo/finalización de la ejecución de tareas. Estos mensajes, junto con las restricciones de tiempo forman la base de la implementación del comportamiento del sistema.

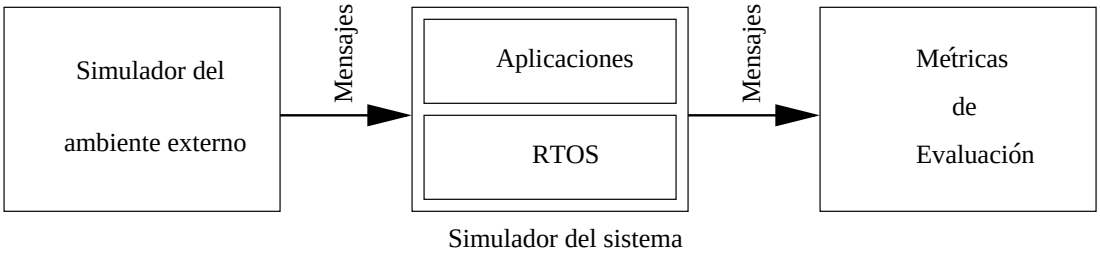

Figura 3.8: Arquitectura básica para la evaluación del sistema simulado

ARTISST cuenta con un conjunto de módulos de entrada inicializados (por medio de alguna función de distribución, o por mensajes generados previamente) que permiten entregar los mensajes para simular el ambiente externo. Se pueden modificar varios módulos para conectarlos con los módulos creados por el usuario para modelar el ambiente externo. Los mensajes de entrada son enviados al simulador del sistema, el cual, los entrega al sistema operativo de tiempo real simulado, que es el encargado de generar los mensajes de salida

relacionados con el sistema operativo de tiempo real y la aplicación. ARTISST proporciona un conjunto de módulos de salida que están a cargo del procesamiento e implementación de los mensajes generados por el simulador. Estos módulos muestran el análisis estadísticos o representaciones gráficas. Se puede desarrollar módulos específicos para evaluar métricas particulares.

3.5.3 Modelo de Tareas en ARTISST

El simulador del sistema de ARTISST identifica una tarea simulada como un flujo secuencial que se ejecuta en el procesador, o se encuentra esperando tiempo de procesador. Es decir, que la tarea ha sido desalojada o bloqueada.

Modelo básico de las tareas. El modelo básico de las tareas en ARTISST es una estructura de datos que define los atributos comunes y que permite extenderse según las necesidades del usuario. Este modelo básico está compuesto de:

- Definición operacional de las tareas.

  El número máximo de tareas concurrentes que se ejecutan. Especifica cuántas tareas asociadas con un modelo de tarea pueden ejecutarse concurrentemente en el simulador.

  Punto de entrada para la tarea. Es una referencia que identifica el tipo de tarea para determinar donde comienza la ejecución.

- Comportamiento temporal de las tareas.

  Características temporales de ejecución de las tareas. Define los tiempos de ejecución de las tareas (para el peor/promedio/mejor caso) y el plazo de respuesta de cada tarea.

  Orden de arribo de las tareas. Verifica la forma en que se activan las tareas del sistema: periódica, esporádica o aperiódicamente. Los primeros dos tipos de arribo se asocian con un parámetro que describe: El período y el tiempo mínimo entre arribos respectivamente.

3.5.4 Estados de las Tareas

Mientras que el modelo permitió definir los atributos abstractos de las tareas. El estado de ejecución de las tarea está asociado al comportamiento de cada tarea, y se encuentra implementado mediante una estructura de datos que se compone de:

Hilo de ejecución. Este mapea a la pila y al contador de programa del *host* donde se lleva acabo la simualción

Modelo de las tareas. Hace referencia la modelo de tarea implementado.

Estado de tiempos para la instancia de la tarea. Consiste en el tiempo de arribo de la tarea y el tiempo de procesador consumido hasta el instante actual.

Monitoreo de ejecución. Consiste de un contador para los eventos que ocurren mientras la tarea se esta ejecutando: Número de desalojos, número de interrupciones del hardware.

3.6 AFTER

AFTER (**A**ssit in **F**ine **T**uning **E**mbedded **R**eal-time systems)[23], es una herramienta para análisis, depuración de errores, y sincronización fina de los plazos de respuesta de sistemas empotrados. AFTER incorpora los esquemas de planificación estáticos y dinámicos, y soporte para threads (tareas) períodicos, servidores aperíodicos, e interrupciones.

El diagrama de AFTER se presenta en la figura 3.9. Las funciones de los módulos se describen a continuación:

3.6.1 Implementación

En el módulo **Implementación** de la figura 3.9(<1>) se define el conjunto de tareas que será analizado por AFTER. En este módulo se requiere definir los puntos de inicio y finalización de las tareas, los cuales deben estar asociados a las especificaciones temporales. Es decir, los puntos de inicio se encuentran asociados a los períodos y los puntos de finalización con los plazos de respuesta.

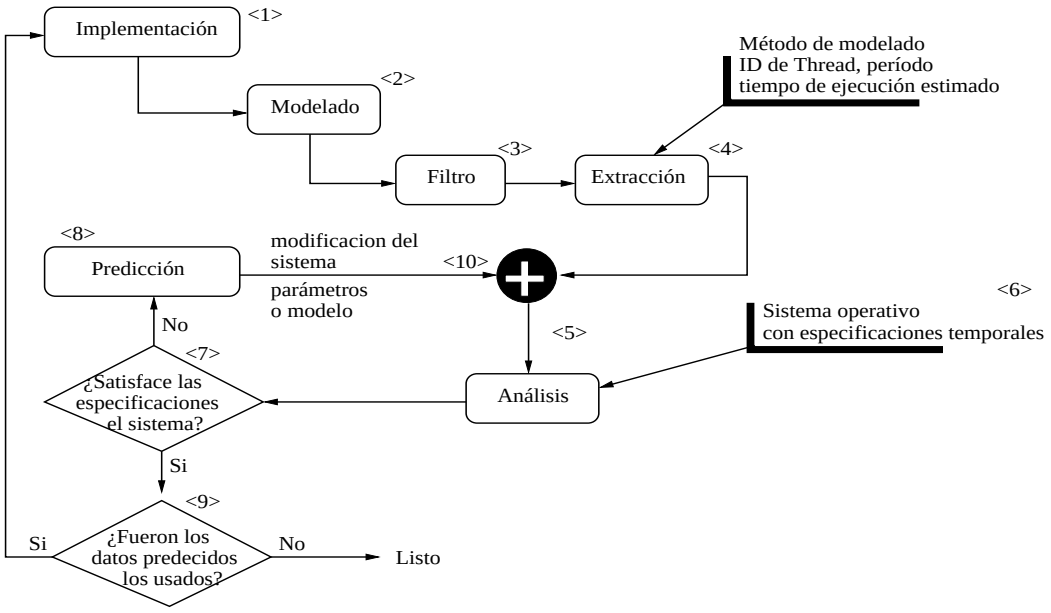

Figura 3.9: Componentes de AFTER

3.6.2 Unidad de Monitoreo

La figura 3.9(<2>) muestra el módulo **Modelado**, también llamado *monitoreo*, el cual contiene una colección de datos y la sincronización del funcionamiento del sistema. Este módulo, cuenta con tres módulos, los cuales son:

- El Módulo monitoreo de hardware. Está hecho de un análizador lógico de procesador, bus del sistema, o puertos de entrada/salida para observar su comportamiento y recabar las estádisticas de su comportamiento sin perturbar el sistema.

- El Módulo monitoreo de software. Está hecho conjuntando las rutinas de medición que registran los eventos que son importantes para el usuario y los cuales pueden ser evaluados por la aplicación.

- El Módulo híbrido. Es una solución que combina los módulos de monitoreo de hardware y software, en el cual usa el análizador lógico mediante una interfaz gráfica.

3.6.3 Unidad de Filtro

La unidad de Filtro se muestra en la figura 3.9(<3>) y tiene como objetivo convertir los datos suministrados por la unidad de monitoreo para que sean leídos por la unidad de Extracción. Los datos que obtiene la unidad de Filtro son: Los identificadores de las tareas, la activación de cada evento, prioridades, y parámetros temporales. La información filtrada es almacenada en un arreglo, para ser procesada por la unidad de Extracción y ser presentada al usuario mediante una ventana gráfica como se muestra en al figura 3.10.

Filter Module

Total CPU utilization 0.76

Thread ID	Type	Priority	Cmin (ms)	Cmax (ms)	Cavg (ms)	Tmin (ms)	Tmax (ms)	Tavg (ms)
01	interrupt	259	0.78	3.84	1.19	8.00	11.00	9.00
99	interrupt	52	0.03	0.03	0.03	16.00	63.00	48.00
03	sporadic	26	1.82	1.94	1.85	83.00	127.00	99.00
05	sporadic	38	0.10	6.66	0.56	10.00	101.00	67.00
06	sporadic	24	0.06	0.06	0.06	79.00	130.00	108.00
0c	sporadic	26	0.36	0.38	0.36	84.00	125.00	99.00
00	sporadic	24	0.08	0.17	0.13	3.00	536.00	93.00
02	periodic	155	0.18	0.20	0.19	15.00	17.00	16.00
04	periodic	12	0.39	0.39	0.39	63.00	552.00	195.00
06	periodic	520	0.29	0.59	0.42	3.00	6.00	4.00
0a	periodic	260	0.43	0.45	0.44	9.00	10.00	10.00
0b	periodic	140	0.14	0.16	0.14	6.00	32.00	16.00
0e	periodic	139	0.12	0.14	0.12	5.00	32.00	16.00

Figura 3.10: Datos procesados por AFTER

3.6.4 Unidad de Extracción

La unidad del Extracción Figura 3.9(<4>) lee la salida de la unidad de Filtro y determina los tiempos de ejecución (mínimo, promedio y peor caso), la frecuencia promedio de cada tarea y las activación mínima entre los arribos de alguna tarea en particular. La unidad Extracción puede proporcionar varias vistas las cuales pueden producir una clase de información que sirve de entrada a la unidad Análisis. Una vista de la unidad de Extracción puede producir el peor tiempo de ejecución de cada tarea, mientras que una segunda vista puede producir el historial de la sincronización de una tarea especifica y una tercer vista puede producir la planificación inmediatamente después que el usuario activa la ejecución de la planificación.

3.6.5 Unidad de Análisis

La unidad del análisis (representado en la Figura 3.9(<5>)) es la base de AFTER y se encarga de realizar el análisis de planificabilidad con los datos proporcionado por la unidad de Extracción. Esta unidad está basada en un sistema de ecuaciones analíticas y está compuesta por dos modos de funcionamiento.

1. Modo de análisis. En cual consiste en realizar el análisis usando los parámetros temporales que fueron dados de alta en la unidad de la Implementación para el sistema empotrado. El análisis se realiza usando un modelo matemático consistente con el algoritmo de planificación.

2. Modo de predicción. Utiliza los parámetros temporales originales del sistema empotrado y los parámetros modificados por el diseñador, para predecir el comportamiento del sistema empotrado usando las nuevas características.

3.6.6 Predicción

Si la unidad de análisis detecta problemas como la pérdida de plazos ó el diseñador desea realizar algunas modificaciones, entonces se activa la unidad de predicción que se muestra en la figura 3.9(<8>).

La unidad de Predicción proporciona dos clases de predicción.

- *Predicción para la modificación de configuración del sistema.* Por ejemplo, cambio del algoritmo de planificación de un estático a un dinámico o viceversa. Para hacer la predicción se utilizan los mismos parámetros temporales pero se usa un modelo diferente para el análisis de planificabilidad.

- *Predicción por modificaciones de parámetros temporales.* Por ejemplo, modificación de parámetros tales como: Tiempos de ejecución y períodos. Para predecir el comportamiento de estos cambios en el sistema empotrado se utiliza el mismo modelo para el análisis de planificabilidad.

Las modificaciones se realizan a través de una interfaz gráfica como la que se muestra en la Figura 3.11. Por ejemplo, si el diseñador desea predecir el efecto de cambiar del algoritmo

Rate monotonic al EDF, puede seleccionar el botón “Preentive EDF”. Asimismo, el diseñador puede determinar el efecto de optimizar uno o más threads (tareas), reduciendo el tiempo de ejecución usado en cada tarea. Una vez realizadas las modificaciones nuevamente la unidad de Análisis realiza los cálculos de planificabilidad (<10>). Si ya no hay problemas (<7>) con el sistema, entonces se tiene el sistema resuelto y puede volver a realizar otra simulación.

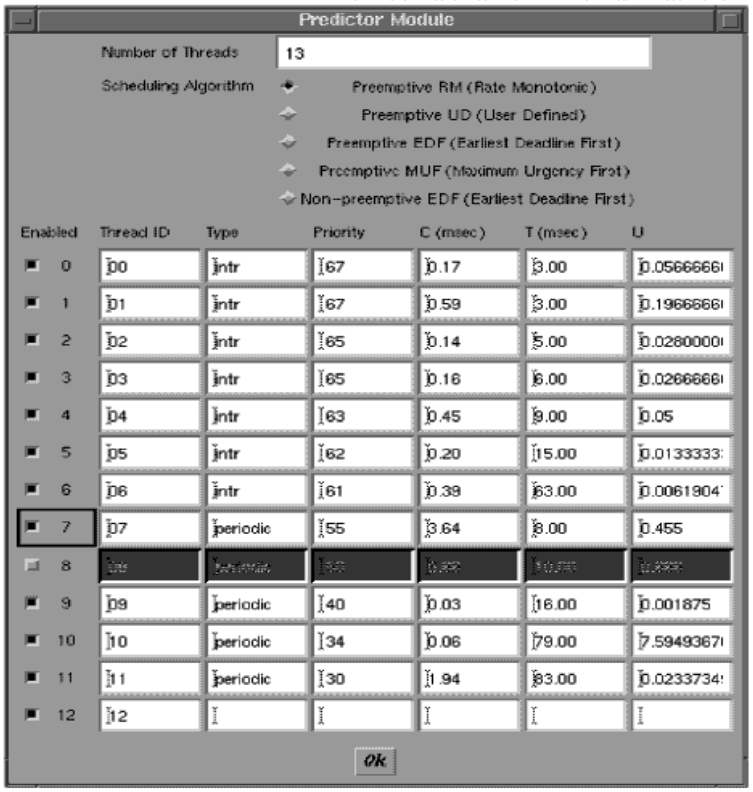

Figura 3.11: Interfaz gráfica: Modificación de parámetros del sistema de tiempo real

Capítulo 4

Herramienta de Simulación: PlataTR

En este Capítulo presentaremos la estructura de la herramienta de simulación PlataTR desarrollada en esta tesis. En la descripción de la estructura identificaremos los módulos lógicos y las funcionalidades de cada módulo de la herramienta. En este capítulo presentaremos el diagrama general de bloques y la arquitectura de la interfaz gráfica de usuario. Además, definiremos las estructuras de datos que permiten la implementación del modelo de tareas que se ha propuesto.

4.1 Motivación

Como se describió en el Capítulo 3, en la actualidad existen una gran variedad de herramientas de simulación para el diseño y análisis de tareas de tiempo real. Estas herramientas tienen la capacidad de modelar un conjunto de tareas con diferentes niveles de complejidad en cuanto a la especificación de tareas. En algunas herramientas, el diseñador de sistemas de tiempo real proporciona un conjunto de tareas mediante instrucciones ó *scripts*, lo que implica tener que conocer todas las instrucciones que se pueden utilizar para la declaración de las tareas y también aprender la sintaxis para poder codificarlas [5, 11, 21, 10]. Otras herramientas cuentan con un generador de tareas [19, 3], donde la generación de las tareas se realiza

de forma pseudo-aleatoria mediante funciones de distribución (Poisson, Uniforme, Normal, etc.). Algunas más cuentan con una interfaz gráfica de usuario lo que las hace más flexibles [19, 21, 10, 9].

La mayoría de las herramientas anteriormente descritas son propietarias y muy complejas, además de que su costo es muy elevado. Por esta razón, en esta tesis decidimos realizar nuestra propia implementación de un simulador para tareas de tiempo real basado en la plataforma de Windows.

La herramienta gráfica desarrollada cumple con las siguientes caracterísitisticas:

1. Presenta la ejecución en forma gráfica de tareas de tiempo real en forma estática y en forma dinámica.

2. Dibuja el comportamiento temporal dinámico de un conjunto de tareas generado y planificado por un Micro-Kernel de tiempo real.

3. Permite modelar la ejecución de un conjunto de tareas bajo distintas políticas de planificación (RM, EDF, LLF, y DM).

4. Se ejecuta bajo la plataforma de Windows de Microsoft.

Esta herramienta gráfica la hemos denominado PlataTR (**Pla**nificador de **Ta**reas de **T**iempo **R**eal). PlataTR ejecuta y simula la ejecución de un conjunto de tareas periódicas de tiempo real, en base a los modelos de planificación Rate Monotonic (RM), Earliest Deadline First (EDF), Least Laxity First (LLF) y Deadline Monotonic(DM), presentados en el Capítulo 2.

4.2 Modelo del Sistema

El problema de la planificación de tareas de tiempo real en un ambiente uniprocesador es complejo debido a las restricciones de tiempo (*deadline*) de las tareas y se complica más debido a que el diseñador de sistemas de tiempo real debe decidir el algoritmo de planificación a usar para diseñar el sistema. De aquí que se requiere de herramientas de diseño que permitan verificar las características temporales del conjunto de tareas de tiempo real.

El modelo de tareas de tiempo real utilizado en PlataTR es el siguiente: se considera un conjunto Γ=$\{\tau_1,\tau_2,...,\ \tau_n\}$ de n tareas periódicas ejecutándose en un solo procesador con las siguientes características:

- Las tareas son independientes. Es decir, no comparten recursos y no tienen restricciones de precedencia.
- El tiempo de arribo de las tareas se realiza en el instante $t \geq 0$.
- Las tareas son expulsables. Es decir, si una tarea τ_i con mayor prioridad solicita tiempo de procesador en un determinado instante de tiempo, la tarea que se este ejecutando en ese instante de tiempo es interrumpida para asignar el procesador a la tarea τ_i.
- Durante la ejecución de las tareas del sistema, el tiempo de cómputo permanece constante (no cambia durante ninguna instancia de ejecución).
- El cambio de contexto no se incluye en la planificación.
- $C_i \leq D_i = T_i$ para cada tarea τ_i.

La tabla 4.1 muestra los parámetros de las tareas utilizados para el desarrollo de PlataTR

Las tareas se ejecutan por varias instancias las cuales el usuario puede observar en un área de dibujo, con la limitante de que esta área de dibujo solo puede mostrar como máximo hasta treinta *ticks* (instantes) de tiempo. El período y el plazo de respuesta de cada tarea (m_APeriodo, m_ADeadline) permanecen constantes durante la ejecución, mientras que m_APeriodoRel y m_ADeadlineRel definen el período y el plazo de respuesta relativo al tiempo que ha transcurrido. Por ejemplo, para las tareas τ_1=(30,10) y τ_2=(50,15) sus plazos de respuestas y sus períodos se incrementan cada vez que termina una instancia de cada tarea. Para la tarea τ_1 el plazo de respuesta absoluto es 30 el cual permance constante durante la simulación, mientras que el plazo de respuesta relativo se incrementa cada vez que termina una instancia; el valor del incremento es la cantidad asignada al plazo de respuesta absoluto (m_ADeadline). Es decir, para la tarea τ_1 sus plazos relativos son 30, 60, 90, 120, etc. De la misma manera será para los períodos de activación. Recordemos que los plazos de respuesta son iguales que los períodos de activación ($D_i = T_i$). La Figura 4.1 muestra los plazos de respuesta relativos para las tareas τ_1 y τ_2.

Parámetro	Descripción
m_ANumTask	Número de tarea
m_ADescription	Descripción de la tarea
m_APlace	Lugar donde se dibuja la ejecución de la tarea
m_APeriodo	Período de activación de la tarea
m_ADeadline	Plazo de respuesta
m_AComputo	Tiempo de cómputo asignado a la tarea
m_AExec	Tiempo que se ha ejecutado la tarea hasta el tiempo actual
m_AUtilizacion	Utilización de cada tarea
m_ADeadlineRel	Plazo de respuesta relativo al tiempo
m_APeriodoRel	Período relativo al tiempo
m_AColor	Color que identifica a la tarea durante la simulación

Tabla 4.1: Parámetros para las tareas de PlataTR.

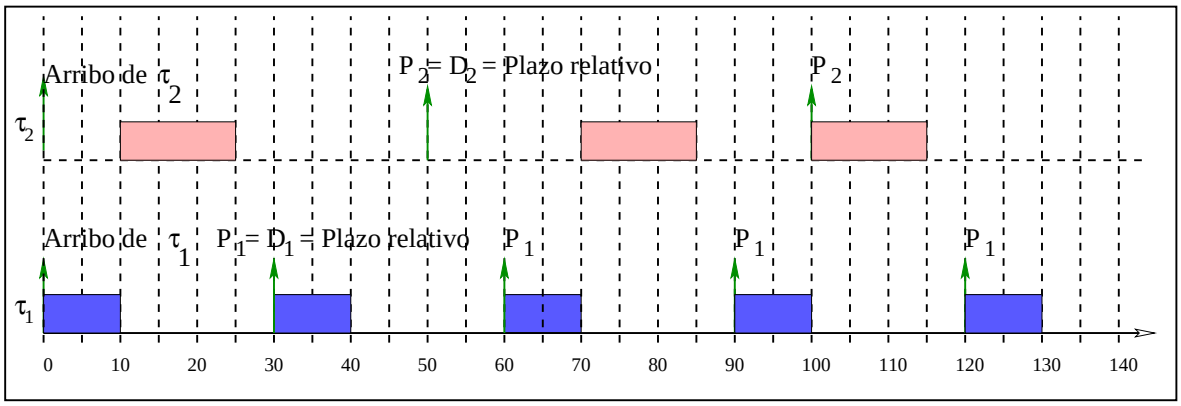

Figura 4.1: Tiempos relativos para las tareas..

4.3 Diagrama General de Bloques

La Figura 4.2 muestra la estructura lógica de PlataTR, la cual se encuentra dividida principalmente en dos partes ó modalidades:

- **Modalidad Estática**

  Consiste en introducir los parámetros de las tareas desde PlataTR, ya sea por medio

de una ventana de diálogo (crear un nuevo conjunto de tareas) o mediante la carga de un archivo de texto.

- **Modalidad Dinámica**

  Consiste en proporcionar un archivo de texto, el cual contiene los eventos generados por la ejecución de un Micro-Kernel de tiempo real durante un intervalo de tiempo. Los eventos son analizados y dibujados por PlataTR con la finalidad de observar visualmente el comportamiento de las tareas.

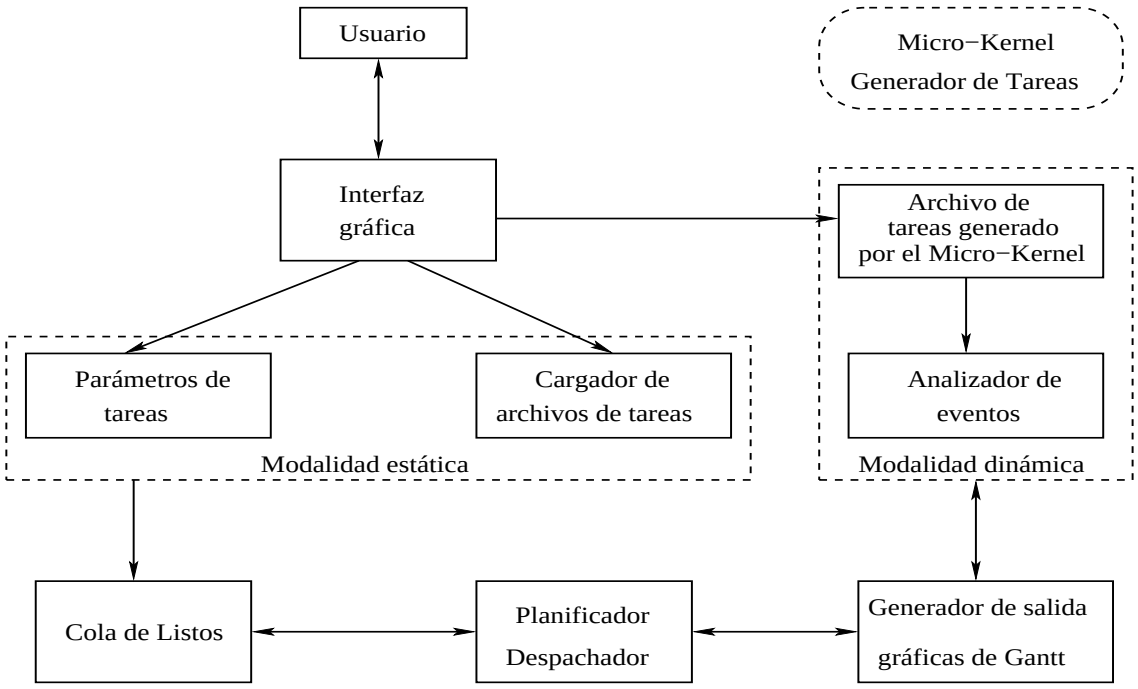

Figura 4.2: Estructura lógica de PlataTR

La descripción por etapas de los módulos de la Figura 4.2 es la siguiente:

1. **Interfaz gráfica de usuario**.

   En la actualidad, un sistema no solo debe ser eficiente y poderoso, sino también, debe ser fácil de usar. Para ello, PlataTR cuenta con un interfaz gráfica amigable, basada en ventanas que aparecen y desaparecen conforme se utilizan. Esto permite que los conocimientos requeridos para el manejo de PlataTR sean minímos.

2. **Módulo de Parámetros de tareas**.

   Este módulo permite al usuario dar como entrada los parámetros temporales de un conjunto de tareas de tiempo real. Del conjunto de tareas, PlataTR despliega en forma gráfica el comportamiento temporal de cada tarea. En el caso de que exista pérdida de plazos de alguna tarea, el módulo indica que existe *sobrecarga* de procesador indicando cual fue la utilización del sistema.

3. **Cargador de archivos de tareas**.

   El Cargador de archivos de las tareas permite leer de un archivo de texto un conjunto de tareas. El archivo a leer, debe tener un formato como el que se presenta en la tabla 4.2.

```
# CINVESTAV-IPN, México, D.F., Miguel Angel Fajardo Ortiz
Task=(1, uno, (0, 0, 255), 1, 0, 30, 30, 10)
Task=(2, dos, (128, 128, 192), 2, 0, 40, 40, 10)
Task=(3, tres, (64, 0, 128), 3, 0, 50, 50, 12)
```

Tabla 4.2: Formato de archivo de tareas de tiempo real creado desde PlataTR.

La primera línea corresponde a un encabezado el cual permite identificar si el archivo puede ser leido por PlataTR y las siguientes líneas corresponden a los parámetros de cada tarea, cada línea correponde a una tarea y cada parámetro se encuentra separado por el caracter ",". La sintaxis es la siguiente:

Task=(*Id*, *Descripción*, *Color*, *Lugar*, *Inicio*, *Período*, *Plazo*, *Cómputo*)

- **Id**. El campo Id corresponde a un identificador númerico de cada tarea y es asignado por PlataTR de acuerdo al orden en que se introducen las tareas.
- **Descripción**. El campo Descripción permite al usuario especificar con letra la actividad que realiza la tarea.
- **Color**. Para el desarrollo de PlataTR hemos utilizado un esquema de colores el cual permite identificar las tareas en el área de dibujo. Windows utiliza un entero de 32 bits para representar un color. El byte menos significativo especifica el rojo,

el siguiente el verde, el siguiente el azul y el siguiente no se utiliza, lo que da lugar a 2^{24} colores. Normalmete a este entero se le denomina *Color RGB*. Windows permite especificar un color mediante la macro **RGB** la cual convierte los colores rojo, verde y azul de 8 bits a un valor de tipo **COLORREF**. La sintaxis de la macro RGB es:

RGB(BYTE *rojo*, BYTE *verde*, BYTE *azul*)

e1 cual es el formato para guardar el color de la tarea en el archivos de texto.

- **Lugar**. El campo Lugar corresponde a la posición que tomará cada tarea dentro del área de dibujo.
- **Inicio**. El campo Inicio indica el instante de tiempo en que cada tarea arriba cuando se incia la ejecución del conjunto de tareas, para el caso de las tareas periódicas todas las tareas arriban en el instante de tiempo cero.
- **Período**. El campo Período corresponde al período de activación de la tarea.
- **Plazo**. El campo Plazo corresponde al Plazo de respuesta asignado a la tarea.
- **Cómputo**. El campo Cómputo indica el tiempo de cómputo asignado a la tarea.

4. **Almacenamiento de tareas**

   El módulo de almacenamiento de tareas, permite guardar un conjunto de tareas a un archivo de texto y se realiza mediante la ventana de diálogo para guardar archivos de Windows como la que se muestra en la Figura 4.3.

5. **Cola de listos**.

   Una vez que se ha cargado un conjunto de tareas, creando un nuevo conjunto o cargando un archivo de tareas, las tareas son enviadas a la cola de listos para ser planificadas.

6. **Módulo de archivo de tareas del Micro-Kernel**.

   Este módulo consiste en cargar un archivo de texto que fue generado por un Micro-Kernel de tiempo real. El archivo es denominado archivo de eventos, debido a que su contenido es una serie de eventos de cada tarea durante el intervalo de tiempo en que el Micro-Kernel se ejecutó.

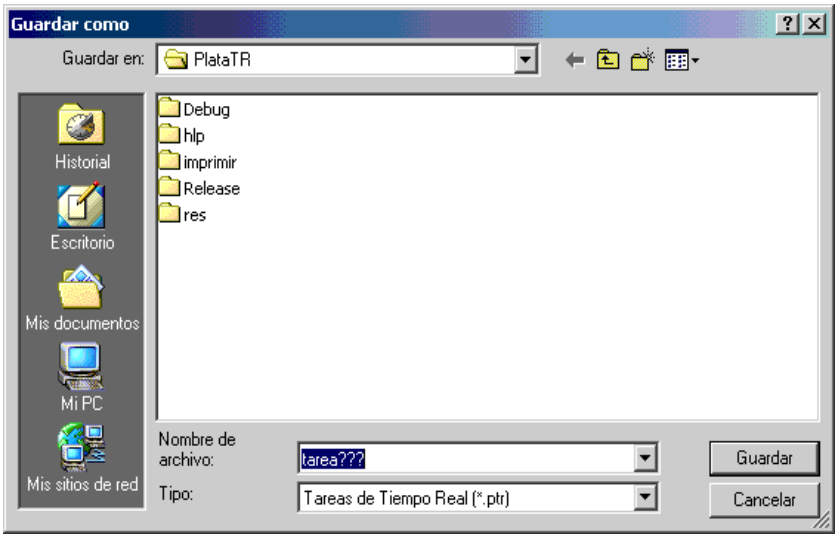

Figura 4.3: Ventana de Windows para almacenar un archivo de tareas en PlataTR.

7. **Simulador de Planificación**.

   El simulador de planificación se encarga de dar una orden de ejecución a las tareas de la cola de listos. Esta orden de ejecución se ejecuta de acuerdo a la política de planificación seleccionada (RM, DM, EDF o LLF).

8. **Analizador de eventos**.

   El analizador de eventos examina el comportamiento de la ejecución de las tareas el cual ha sido generado por el Micro-Kernel de tiempo real. Las tareas son enviadas a una cola de eventos, apartir de donde se graficará el comportamiento que tuvieron al ser planificadas por el Micro-Kernel. Los eventos que se analizan son: los períodos, la pérdida de plazos y la finalización de las tareas.

9. **Generador de salida**

   Una vez que la tarea τ_i ha sido seleccionada mediante el algoritmo de planificación para ser ejecutada, se envía al generador de salida cuya funcionalidad consiste en realizar la operaciones indicadas (calcular las coordenadas (x,y) donde será dibujada la tarea τ_i en el área de dibujo) para presentar visualmente en pantalla la gráfica de Gantt al usuario.

10. **Micro-Kernel**

    Aunque el desarrollo del Micro-Kernel de tiempo real [24], no fue parte de esta tesis, mencionamos algunas de sus caracteríticas:

    El Micro-Kernel tiene la capacidad de ser instalado en sistemas empotrados que contienen poca memoria (32 Kb).

    Los servicios básicos con los cuales cuenta el Micro-Kernel son:

    - **Manejador de procesos (tareas)**. El manejador de procesos ofrece varias funciones de soporte tales como la creación y eliminación de tareas, la planificación de tareas mediante los algoritmos de planificación RM, EDF y FIFO Round Robin y el cambio de contexto. Así mismo, el Micro-Kernel cuenta con la función de retrasar a un proceso. De esta forma es posible dormir a un proceso por un tiempo determinado.
    - **Manejador de interrupciones**. El manejador de interrupciones controla la solicitudes de interrupción del teclado y el reloj.
    - **Sincronización de procesos**. La sincronización de procesos es otro servicio básico que proporciona el Micro-Kernel de tiempo real el cual es utilizado para sincronizar y comunicar procesos mediante semáforos y buzones.

4.4 Estructura de la Interfaz Gráfica de Usuario

La interfaz gráfica de usuario de PlataTR, consiste de una ventana principal, ventanas de diálogo cuyo objetivo es proporcionar la interacción con el usuario, y el área de dibujo donde se visualiza la ejecución de las tareas mediante gráficas de Gantt.

Para el desarrollo de la interfaz gráfica de PlataTR hemos usado las bibliotecas de MFC (Microsoft Foundation Class) [17, 8] las cuales tienen la funcionalidad de proporcionar las operaciones de creación y manipulación de ventanas para Windows. Una característica de las MFC's en la construcción de aplicaciones es que proporcionan un marco genérico para gestionar el almacenamiento y vizualización de los datos. Las MFC's utilizan una arquitectura denominada *documento-vista* (Document/View Arquitecture). Los objetos que intervienen en

la arquitectura documento-vista se muestran en la Figura 4.4. Estos objetos son: *Aplicación, Plantilla de documento, Documento y Vista*, los cuales responden de forma conjunta a eventos que el usuario genera, y se comunican entre sí por medio de mensajes. Estos objetos tienen las siguientes funcionalidades:

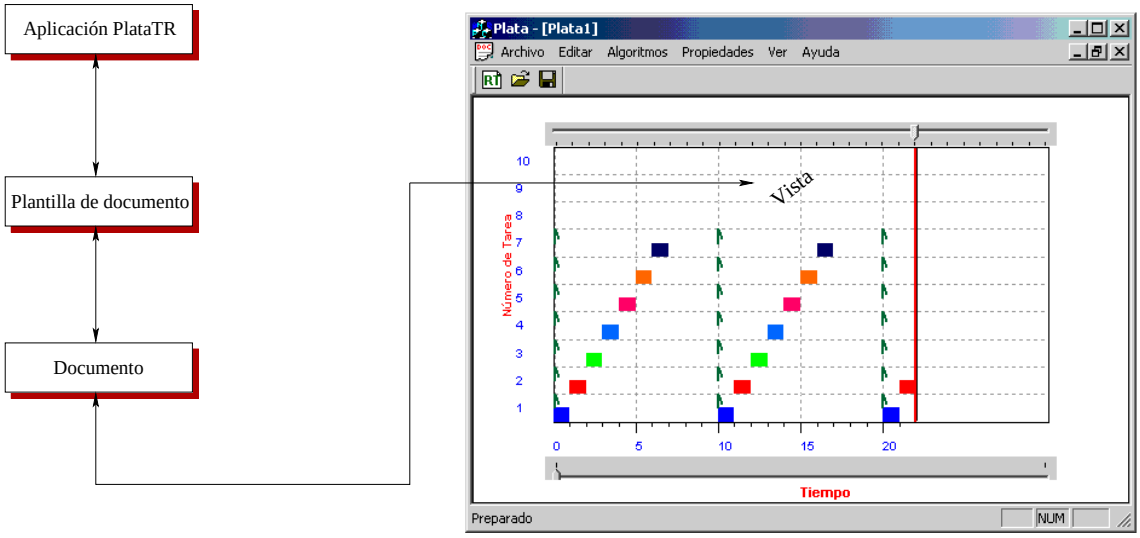

Figura 4.4: Arquitectura documento-vista de PlataTR

- **Aplicación (PlataTR)**.

  Tiene como objetivo iniciar, ejecutar, y finalizar PlataTR, Además crea y gestiona la plantilla de documento.

- **Plantilla de documento**.

  Es un mecanismo para integrar documentos, vistas y ventanas. Asocia el objeto documento con la vista donde se visualizarán los datos.

- **Documento**.

  Almacena la funcionalidad necesaria para la administración de los archivos: Abrir o guardar archivos.

- **Vista**.

  Es la interfaz entre el usuario y los datos del objeto documento. Por lo que es con este

objeto con el que se determina cómo se visualizan los datos y como puede actuar el usuario sobre ellos.

4.4.1 Ventana Principal

La ventana principal es una instancia de la clase **CWnd** de las MFC y es muy común a una ventana típica de Windows. La ventana principal de PlataTR se presenta en la Figura 4.5 y la respectiva numeración identifica los siguientes objetos.

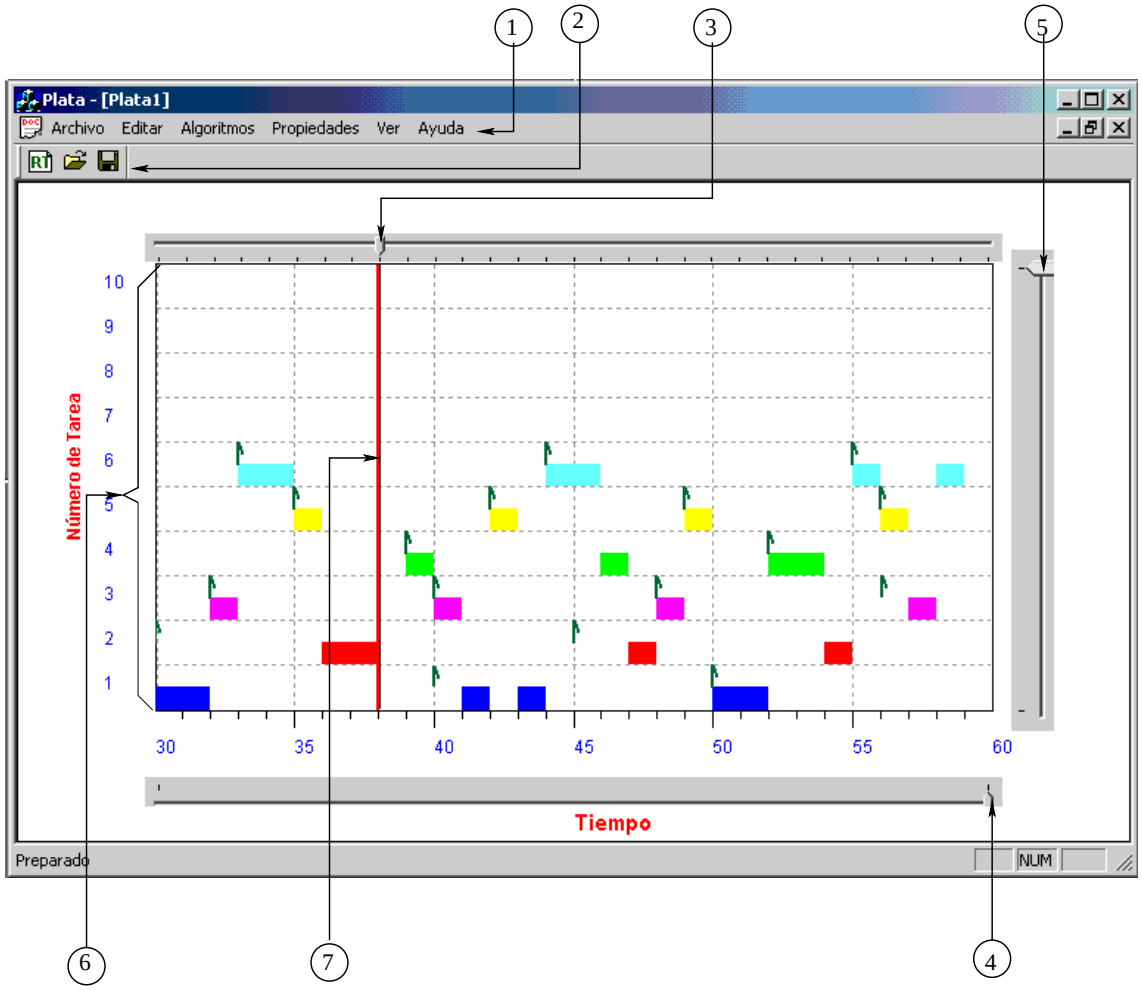

Figura 4.5: Ventana principal de PlataTR

1. **Barra de menús**.

   Visualiza un conjunto de comandos disponibles para el uso de PlataTR. La Figura 4.6

muestra los comandos de menú con los que cuenta PlataTR los cuales han sido divididas en dos grupos los comandos que son obligatorios para la introducción de un conjunto de tareas y su ejecución y los comandos que pueden o no ser utilizados. A continuación se hace una descripción de los comandos utilizados en la barra de menús.

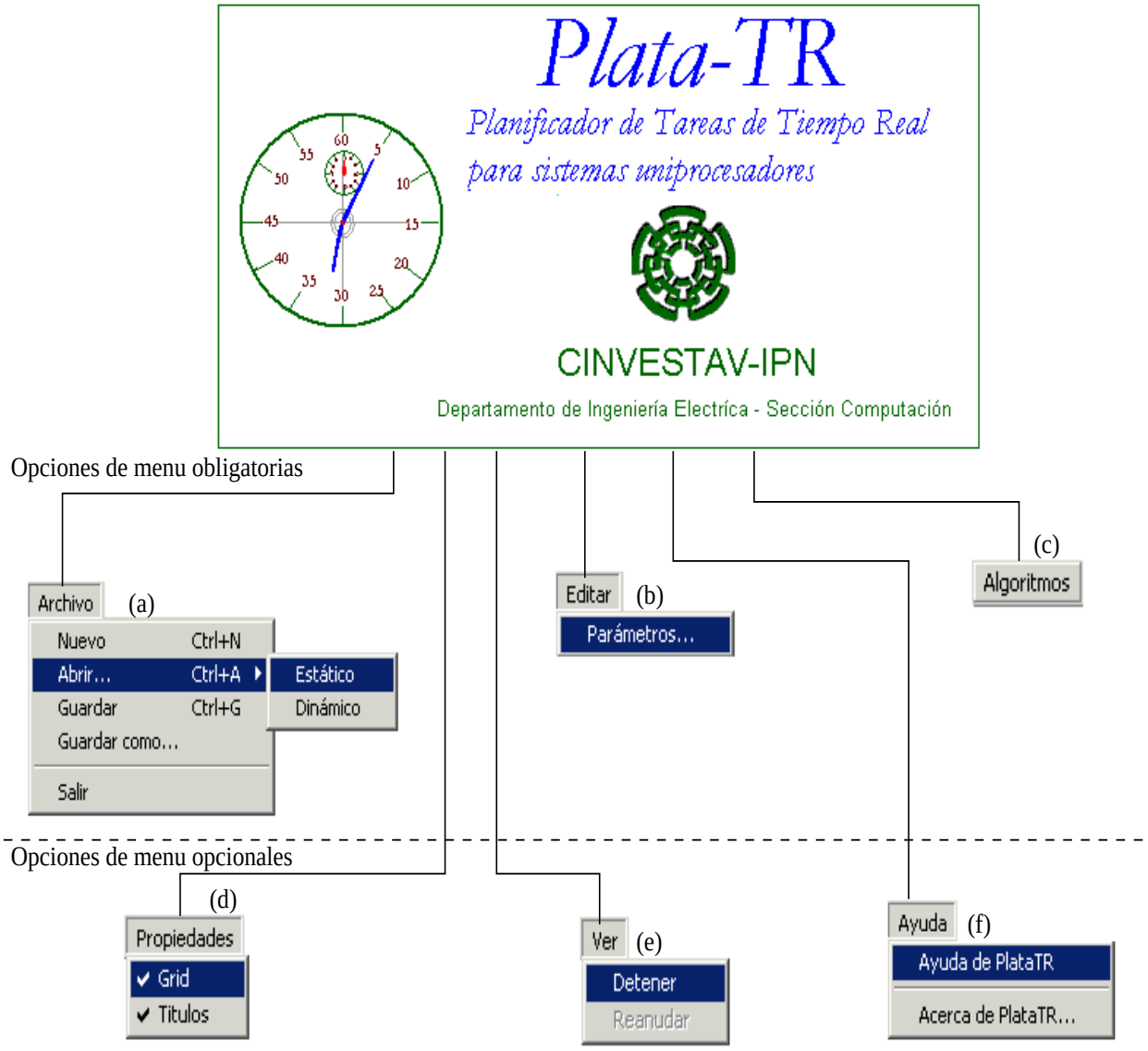

Figura 4.6: Opciones de menú de PlataTR

Archivo. La Figura 4.6(a) muestra esta opción la cual esta formada de los siguientes comandos:

- *Nuevo*. Con éste comando, se introduce un nuevo conjunto de tareas.
- *Abrir*. Se vincula con un submenú, para abrir un archivo de tareas, si el archivo fue creado con PlataTR se selecciona el comando *Estático* o el comando *Dinámico* si el archivo de tareas fue generado por el Micro-Kernel.
- *Guardar y Guardar como*. Guarda los parámetros de las tareas que se tienen en memoria.
- *Salir*. Este comando permite finalizar la sesión de PlataTR.

Editar. La Figura 4.6(b) muestra el comando Editar, el cual se encuentra vinculado con una ventana de diálogo (Figura 4.8) que permite consultar y modificar las tareas del sistema y sus parámetros temporales.

Algoritmos. Presenta los algoritmos que el usuario puede seleccionar mediante una ventana de diálogo; este comando se muestra en la Figura 4.6(c).

Propiedades. La Figura 4.6(d) muestra este comando. Como puede observarse cuenta con dos comandos *Grid* y *Titulos* las cuales si se encuentran activadas muestran una cuadrícula gris y los títulos de los ejes x y y del área de dibujo.

Ver. Este comando cuenta con los comandos *Detener* y *Reanudar*. Permite parar la ejecución y continuar con la ejecución de la planificación de las tareas. La Figura 4.6(e) muestra estos comandos.

Ayuda. La Figura 4.6 muestra los comandos *Ayuda de PlataTR* y *Acerca de PlataTR* proporcionan información acerca del manejo de PlataTR, la versión, y los autores que han realizando la herramienta.

2. **Barra de herramientas**.

   La barra de herramientas contiene botones gráficos los cuales al hacer click en algún bóton produce la ejecución de una de los comandos de la opción Archivo. Esto botones gráficos corresponden a los comandos *Nuevo, Abrir y Guardar* un conjunto de tareas para la modalidad estática.

3. **Botón de desplazamiento para ticks**.

   Este botón permite ubicarnos en el tick específico para verificar la tarea que se ejecutó en el instante donde se coloca el botón.

4. **Botón de desplazamiento para control de página horizontal**.

   Permite desplazarse a una página anterior para verificar como se ejecutaron las tareas.

5. **Botón de desplazamiento para el control de página vertical**

   Permite desplazarse sobre el eje y para observar la planificación de más de diez las tareas.

6. **Área de dibujo**.

   Esta área de la vista es donde se dibuja el comportamiento de las tareas de acuerdo al algoritmos de planificación seleccionado. El eje x muestra el tiempo de ejecución y el eje y muestra el número de la tarea.

7. **Línea de ploteo**.

   La línea de ploteo señala la tarea que se esta ejecutando o la tarea que se ejecutó en la ubicación donde se encuentra. Esta línea se encuentra sincronizada con el bóton de desplazamiento para ticks.

4.4.2 Ventanas de Diálogo

Las ventanas de diálogo son uno de los elementos más utilizados para la interacción de PlataTR con el usuario. De manera general, un diálogo se define como una ventana especial que contiene en su interior ventanas hijas que son controles (botones, listas, cajas de edición, listas desplegables). Cuando una ventana de diálogo se despliega puede habilitarse de forma *modal* o *no modal*. La forma *modal* permite que una ventana se habilite en forma exclusiva, lo cual indica que deja inhabilitada cualquier operación con el resto de la aplicación. Este modo de ventana tiene que ser cerrado para continuar con la operación de la aplicación (normalmente pulsando el botón *Aceptar* o el botón *Cancelar*). Por otro lado, en la forma *no-modal*, la ventana de diálogo de la aplicación es desplegada y permite acceder a otros puntos de la aplicación en cualquier momento.

En PlataTR se utiliza el tipo de ventana de diálogo modal.

La clase *CDialog*, la cual esta derivada de CWnd, encapsula las funcionalidades básicas para las ventanas de diálogo. Entre las más importantes se tienen:

- El constructor de la clase *CDialog::CDialog*, el cual requiere de dos parámetros: el nombre de la plantilla de diálogo que utilizará y una referencia a la ventana padre del diálogo.
- La función *CDialog::DoModal* aplicada sobre un objeto construido previamente, permite visualizarlo en modo modal.
- La función *CDialog::Create* aplicada sobre un objeto construido previamente, permite crear diálogos no modales. Estos diálogos finalizan con la función heredada de CWnd DestroyWindow().
- La función *CDialog::OnInitdialog* permite inicializar las variables de la ventana de diálogo.
- *CDialog::OnOk*/*CDialog::OnCancel* permiten realizar las operaciones de aceptación o cancelación del objetivo del diálogo.

A continuación se presentan las ventanas de diálogo con las que el usuario puede interactuar en PlataTR.

La Figura 4.7 presenta la ventana de diálogo *Parámetros de tareas* con la cual podemos crear un nuevo conjunto de tareas, esta ventana es activada en el momento de seleccionar la opción *Archivo->nuevo*. Una característica de esta ventana se presenta en el botón que corresponde a la etiqueta *Color para la tarea*, debido a que se encuentra vinculado con la ventana Color de Windows.

La ventana Color permite al diseñador seleccionar un color de la paleta de colores de Windows para ser asignado al dibujo de una tarea. Este color será útil para identificar a la tarea durante su ejecución.

La Figura 4.8 presenta la ventana de diálogo denominada *Conjunto de tareas*, la cual permite consultar y modificar los parámetros de las tareas con las que se esta trabajado. Además, permite insertar o eliminar una tarea del sistema. Esta ventana de diálogo se

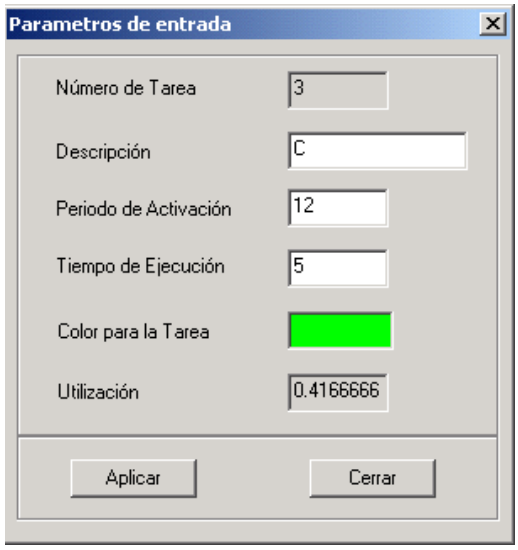

Figura 4.7: Parámetros de entrada de un conjunto nuevo de tareas

activa cuando se selecciona el comando *Editar* de la Barra de Menús. La rejilla en la ventana *Conjunto de Tareas* (de la Figura 4.8) organiza y controla una colección de celdas conteniendo los parámetros de las tareas.

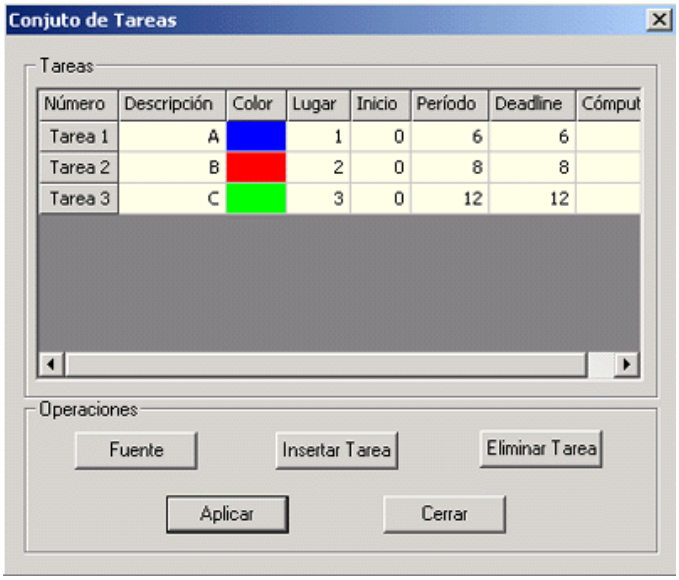

Figura 4.8: Tabla de edición de las tareas

La Figura 4.9 muestra la ventana de diálogo *Algoritmos de Planificación*, la cual incluye

los algoritmos que se pueden aplicar a un conjunto de tareas en PlataTR. Es importante destacar que dada la modularidad con la que se realizó la herramienta PlataTR, es posible añadir nuevos algoritmos de planificación sin muchas complicaciones. En el Capítulo 5, se indicará la forma de incluir nuevos algoritmos de planificación a PlataTR.

La ventana Algoritmos de Planificación se activa cuando el diseñador selecciona la opción *Algoritmos* de la Barra de Menús.

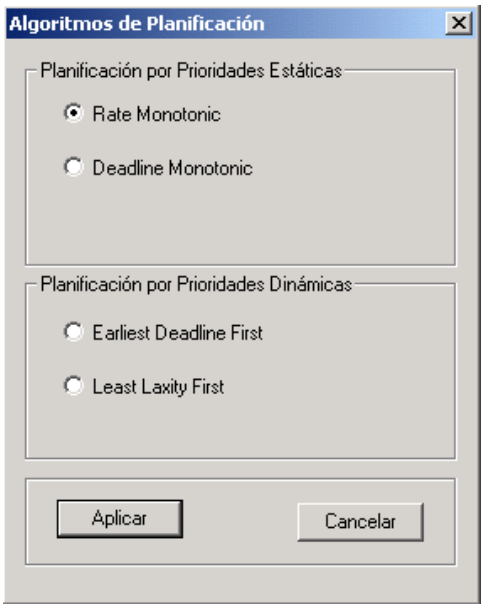

Figura 4.9: Seleccionar el algoritmo de planificación de se desea aplicar al conjunto de tareas

4.5 Metodología para la Planificación de un Conjunto de Tareas

En esta sección se establece la metodología para que un usuario pueda realizar el análisis y simulación de un conjunto de tareas de tiempo real en PlataTR. Como se mencionó en la Sección 4.3 existen dos formas de proporcionar un conjunto de tareas de tiempo real a PlataTR (modalidad estática y modalidad dinámica). La Figura 4.10 muestra el diagrama de flujo que establece como se realiza la simulación de un conjunto de tareas de tiempo real en PlataTR.

En la *modalidad estática*, es posible generar un conjunto de tareas de tiempo real de dos formas. La primera, consiste en dar de alta un nuevo conjunto de tareas proporcionando los

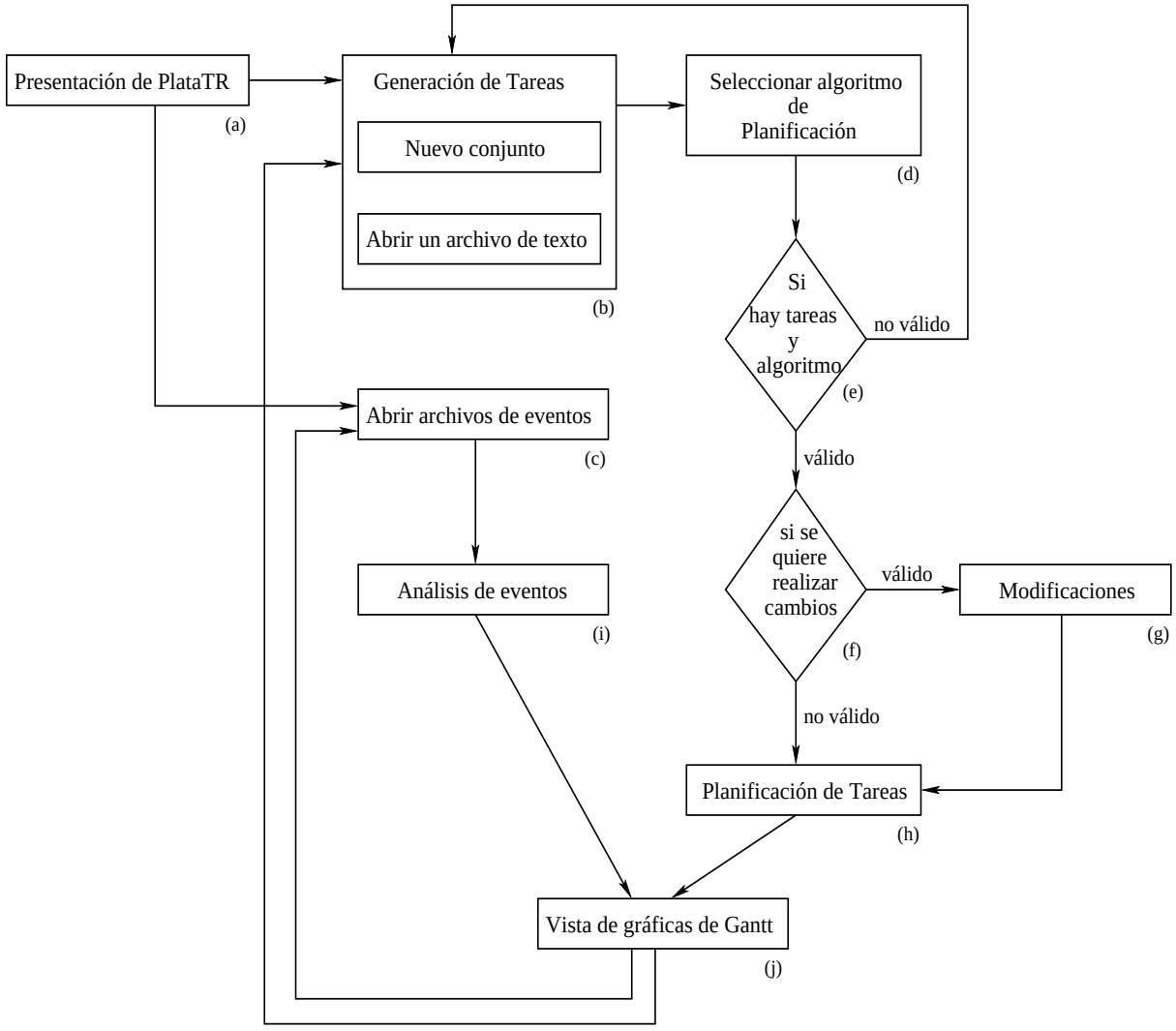

Figura 4.10: Metodología de diseño y análisis de un conjunto de tareas de tiempo real

parámetros temporales de cada tarea, y la segunda forma consiste en cargar un archivo de texto el cual contenga los parámetros temporales de cada tarea (el cual halla sido alamacenado desde PlataTR en alguna sesión anterior). Una vez que PlataTR cuenta con un conjunto de tareas, estas son enviadas ha una estructura dinámica denominada cola de listos (ver Figura 4.10(b)). Adicionalmente, en PlataTR es necesario seleccionar el algoritmo de planificación para iniciar con la simulación de las tareas. Mediante esta simulación será posible examinar el comportamiento de las tareas (ver Figura 4.10(d)). Una vez que se tiene un conjunto de tareas y el algoritmo de planificación es posible entonces iniciar la ejecución de la planifiación

de las tareas (ver Figura 4.10(e)). El diseñador de un sistema de tiempo real cuenta con la facilidad que le proporciona PlataTR de hacer modificaciones a un conjunto de tareas ya sea modificando los parámetros temporales, insertando o eliminado una tarea (ver Figura 4.10(g)). Una vez que se ha proporcionado el algoritmo de planificación y el conjunto de tareas o se hicieron las modificaciones que el diseñador de sistemas de tiempo real crea pertinentes, PlataTR simula la ejecución de las tareas de acuerdo al algoritmo de planificación seleccionado (ver Figura 4.10(h)). Cada tarea simulada es dibujada en PlataTR y pasa por un análisis para calcular la posición que tomará en el área de dibujo de PlataTR Figura 4.10(j).

Por otro lado, en la *modalidad dinámica*, a PlataTR lee un archivo de texto (proporcionado por el usuario), el cual contiene los eventos generados por la ejecución del Micro-Kernel de tiempo real (ver Figura 4.10(c)). En esta modalidad, en cada instante de tiempo se verifica el evento que el Micro-Kernel de tiempo real generó (inicio de ejecución de alguna tarea, pérdida de plazo) (ver Figura 4.10(i)). Una vez que se ha comprobado el evento generado por el Micro-Kernel, el siguiente paso que realiza PlataTR consiste en calcular las coordenadas gráficas mediante las cuales permitirán mostrar cada evento en el área de dibujo (ver Figura 4.10(j)).

Una vez que el diseñador del sistema (el usuario) ha verificado el comportamiento de un conjunto de tareas de tiempo real, será capaz de decidir si es necesario cambiar los parametros de las tareas o si es necesario seleccionar un nuevo algoritmo de planificación (ver Figuras 4.10(f) y 4.10(g)).

Capítulo 5

Implementación y Ejemplos

En el capítulo anterior se examinó la arquitectura y la estructura lógica de la herramienta PlataTR. En este capítulo se examinará a detalle la implementación de la herramienta.

Debido a que la herramienta PlataTR se desarrolló en lenguaje Visual C++, en este capítulo presentaremos la implementación de las clases de objetos desarrolladas. Además, presentaremos algunos ejemplos de planificación que la herramienta permite graficar. En estos ejemplos se utilizarán diferentes conjuntos de tareas periódicas utilizando los algoritmos de planificación RM, DM, EDF y LLF. Además se mostrará la ejecución gráfica de los eventos generados por el Micro-Kernel de tiempo real.

5.1 Implementación de los eventos de tiempo real

La parte fundamental que compone a la herramienta PlataTR es el graficador de la ejecución de las tareas de tiempo real. La graficación de la ejecución de las tareas, se realiza trazando sobre la pantalla su correspondiente gráfica de Gantt. Esta gráfica se presenta en una ventana móvil de forma dinámica. En otra palabras la gráfica de Gantt se presenta en una ventana corrediza.

Para el trazo de cada evento de ejecución de las tareas, se utiliza el reloj. Por tratarse de

una simulación, hemos utilizado la función SetTimer() del temporizador de Windows para definir la frecuencia del reloj. Esta frecuencia nos permite definir los eventos temporales de cada tarea de tiempo real, como son la activación, el inicio y la finalización de la ejecución, y el plazo de respuesta. SetTimer() cuenta con un parámetro con el cual se especifica el intervalo de tiempo (en milisegundos) que tiene que transcurrir para que se ejecute la función OnTimer(). OnTimer() permite especificar las acciones que se ejecutarán periódicamente. De esta forma, SetTimer() ayuda a la ejecución de los eventos periódicos (tareas periódicas) que se requieren simular en la herramienta. SetTimer() genera un tick de reloj cada 54.925 milisegundos. Dado que esta es una frecuencia muy alta, la animación y la ejecución de las tareas no puede ser observada a detalle por el diseñador del sistema de tiempo real. Por esta razón, hemos re-definido el período de generación de ticks de manera que se permita observar la ejecución de las tareas. Para efectos de la simulación hemos definido la activación de OnTimer() cada 250 milisegundos.

5.2 Clases implementadas en la herramienta

La Figura 5.1 muestra la jerarquía de clases implementadas en la herramienta PlataTR. mientras que la Figura 5.2 muestra los objetos visuales que corresponden a las clases implementadas en la herramienta PlataTR.

5.2.1 Clase CPlataView

La primera clase que examinaremos es *CPlataView*, la cual proporciona la funcionalidad necesaria para la visualización de la vista principal de PlataTR. CPlataView es una clase derivada del manejador de ventanas de Windows (CView). La Figura 5.2.1 muestra la ventana (objeto) que manipula la clase CPlataView.

Las funciones implementadas en esta clase son:

- **OnTimer()**. Esta función emula el funcionamiento del reloj para la planificación de las tareas de tiempo real. PlataTR realiza un tick de reloj en base a la función SetTimer() y genera un evento de reloj cada 250 milisegundos, lo que origina que se ejecute la función OnTimer().

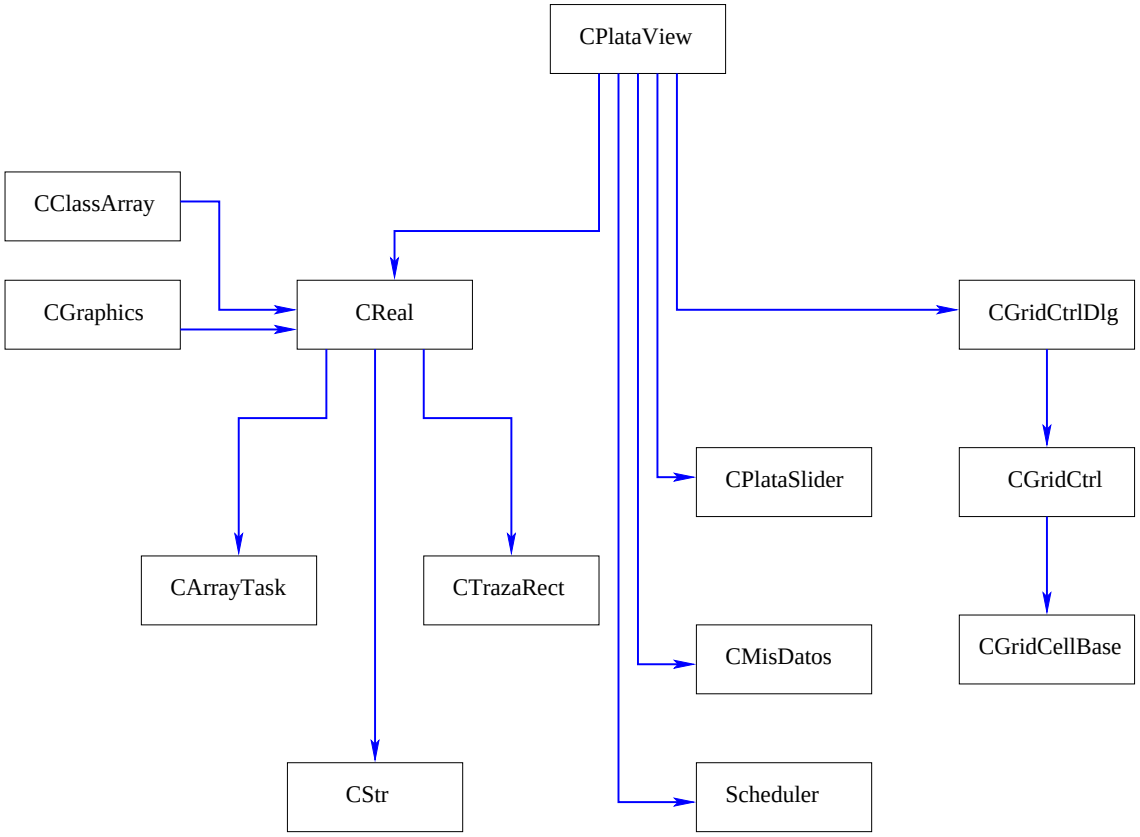

Figura 5.1: Jerarquía de Clases de PlataTR

- **OnInitialUpdate()**. Esta función es invocada por PlataTR antes de que la ventana principal sea mostrada por primera vez. Con esta función se establecen los objetos que se deben mostrar en la ventana principal. La función OnInitialUpdate() invoca la función *InitialSlider()* para mostrar los botones de desplazamiento de PlataTR y activa la función SetTimer() de Windows.

- **OnDraw()**. Esta función permite redibujar la vista de PlataTR cuando la ventana principal de PlataTR se ha maximizado, o bien cuando PlataTR pasa a ocupar el primer plano del escritorio de Windows.

- **TickRange()**. Esta función permite dibujar el intervalo de ticks en el área de dibujo. Esta función lleva el control del número de ticks que son observados en el área de dibujo.

- **InitialSlider()**. Esta función establece la posición de los botones de desplazamiento

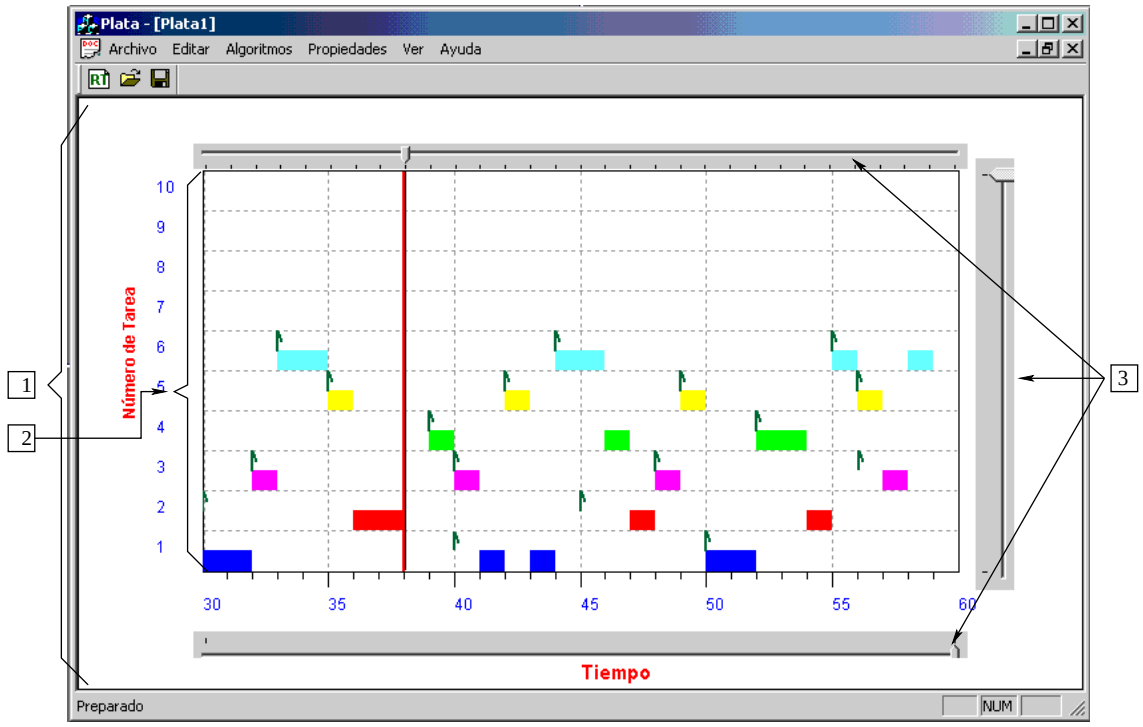

Figura 5.2: Elementos visuales de las Clases de Objetos de PlataTR

mostrados en la ventana principal de PlataTR. La posición de los botones de desplazamiento varia de acuerdo al tamaño de la ventana principal de PlataTR, es decir, esta varía dependiendo de si se inició con la ventana principal de PlataTR en el modo de tamaño maximizada o tamaño normal.

- **ResetSlider()**. Esta función se encarga de reiniciar los valores de los botones de desplazamientos, con el fín de dar inicio a una nueva ejecución.

- **OnViewDetener()**. Esta función se encuentra vinculada con el comando de la barra de menús *Ver->Detener* la cual tiene el objetivo de detener la ejecución de un conjunto de tareas.

- **OnViewReanudar()**. De la misma forma que la función OnViewDetener() la función OnViewReanudar() se encuentra vinculada con la barra de menús de PlataTR. Esta funcion permite continuar con la ejecución de las tareas, después de haber sido suspendida con el comando *Ver->Detener*. La función OnViewReanudar() se encuentra

vinculada con el comando *Ver->Reanudar*.

- **GetYValues()**. Esta función GetYValues() permite obtener los valores de la posición en el área de dibujo, en donde se graficará cada tarea.

- **MoveWindows()**. Esta función es la encargada de realizar la actualización de las coordenadas gráficas de los objetos que contiene la vista (botones de desplazamiento, títulos de los ejes (x,y), y área de dibujo) cuando el usuario mueve (desliza) la ventana principal en la pantalla.

5.2.2 Clase CGraphics

Windows dibuja los gráficos (incluyendo los textos) utilizando la interfaz de dispositivos gráficos GDI, (*Graphics Device Interface*), la cual se encarga de llamar a las rutinas de los distintos manejadores de dispositivo (*drivers* de video, de impresora y de trazadores gráficos) que son los que directamente actúan sobre el dispositivo. La clase *CGraphics* calcula las coordenadas de cada uno de los objetos utilizados en el área de dibujo de PlataTR. La Figura 5.2.2 muestra los objetos que se implementan en la clase CGraphics. Las funciones vinculadas a la clase CGraphics son:

- **SetPrintScale()**. Obtiene el ancho y el alto de pixeles de la ventana PlataTR.

- **RecalcRects()**. Calcula los margenes del área de dibujo.

- **Title()**. Asigna el tipo de fuente (letra) que se utiliza para mostrar las etiquetas de los ejes (x,y).

- **XAxisTitle()**. Efectua las operaciones con las que se manipula una cadena de caracteres para que pueda ser dibujada en la vista de PlataTR. Esta función realiza las operaciones para que las etiquetas se visualizen paralelamente al eje x del área de dibujo.

- **YAxisTitle()**. De la misma manera que la función XAxistitle(), la función YAxisTitle() establece la forma en como se debe visualizar una cadena de caracteres en la vista de PlataTR. La diferencia radica en que la visualización gráfica se hace dando un giro de 90 grados a la etiqueta con respecto al eje x.

- **Grid()**. Dibuja la rejilla que se muestra en el área de dibujo, lo que permite tener una mayor claridad en la visualización.

5.2.3 Clase CClassArray

La clase CClassArray proporciona una variedad de funciones que permiten manipular los arreglos definidos en PlataTR. Estas funciones se encuentran clasificadas de la siguiente forma:

Límites.

- **GetSize**. Devuelve el número de elementos de un arreglo.
- **GetUpperBound**. Devuelve el índice superior de un arreglo. Donde el índice inferior es cero y el superior es GetSize-1.
- **SetSize**. Fija el número de elementos de un arreglo vacío o existente.

Liberar memoria.

- **FreeExtra**. Libera la memoria no utilizada por encima del límite superior actual.
- **RemoveAll**. Libera toda la memoria ocupada por los elementos del arreglo.

Acceso a elementos del arreglo.

- **GetAt**. Devuelve el elemento del arreglo especificado.
- **SetAt**. Sustituye el elemento del arreglo especificado. El índice especifica la posición del elemento el cual debe estar dentro de los límites del arreglo.

Insertar o eliminar elementos.

- **Add**. Añade una nueva tarea al final del arreglo.
- **InsertAt**. Inserta una tarea en la posición indicada.
- **RemoveAt**. Borra una o más tareas del arreglo a partir de la posición indicada.

5.2.4 Clase CScheduler

En la clase *CScheduler* se implementa la ventana de diálogo *Algoritmos de Planificación*. Esta clase permite seleccionar el algoritmo de planificación que se desea aplicar mediante Botones de Selección. Como se mencionó en el Capítulo 5, PlataTR se ha desarrollado de manera que no presente dificultades para incluir un nuevo algoritmo. La clase *CSheduler* permite añadir un botón de selección nuevo, el cual representará el nuevo algoritmo de planificación

Para añadir un nuevo algoritmo de planificación, el entorno de desarrollo de Visual C++ 6.0 (Figura 5.3) cuenta con una ventana denominada *Workspace* (Figura 5.3.1). En la parte inferior de dicha ventana se visualizan tres opciones (Figura 5.3.2) que dan acceso a los componentes de PlataTR. Estas opciones son:

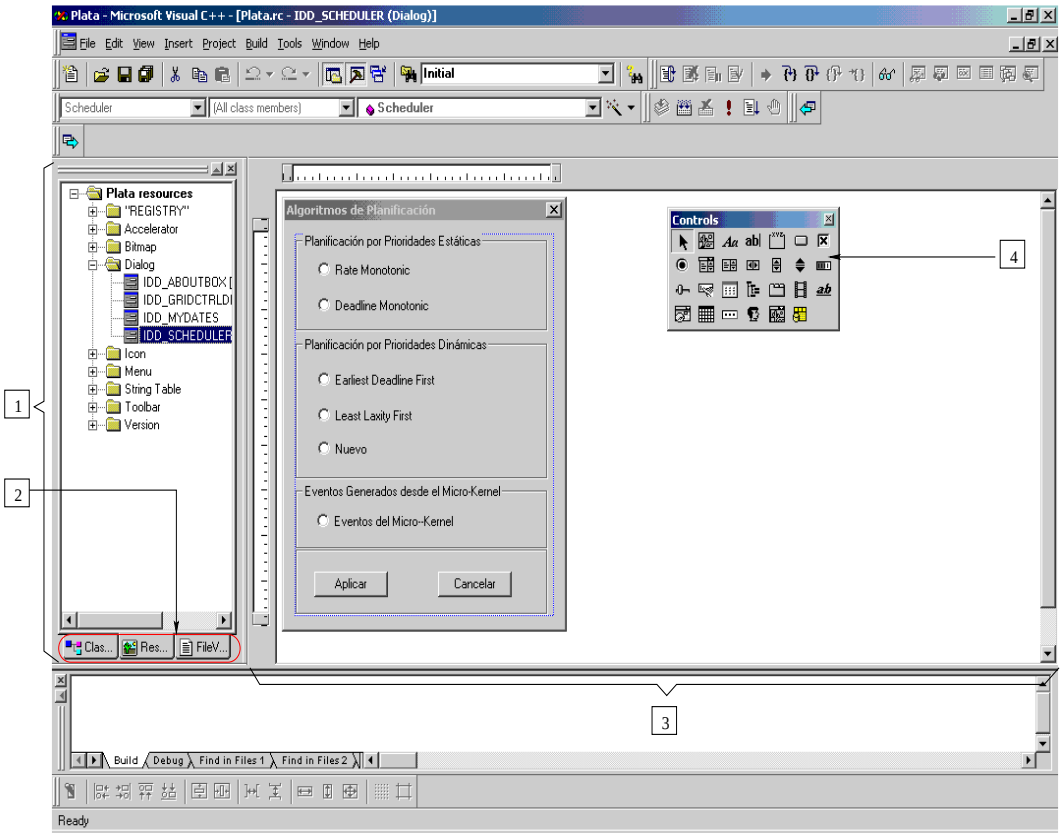

Figura 5.3: Añadiendo un nuevo algoritmo de planificación en PlataTR desde Visual C++ 6.0

- *ClassView*. Muestra las clases que componen la aplicación herramienta PlataTR.

- *ResourceView*. Muestra los recursos que forman parte de PlataTR (Ventanas de diálogo, menús, mapa de bits, iconos y barra de herramientas).

- *FileView*. Muestra los nombres de los archivos fuente que componen la herramienta PlataTR.

Para insertar un nuevo algoritmo de planificación, es necesario seleccionar la opción *ResouceView* de la ventana *Workspace* (Figura 5.3.2). La cual despliega una jerarquía de directorios como la que se muestra en la Figura 5.3.1. Los algoritmos de planificación se encuentran agrupados en la ventana de diálogo denominada *Algoritmos de Planificación* la cual se encuentra almacenada dentro del directorio *Dialog*. Por lo tanto, se debe elegir el directorio *Dialog* y seleccionar el identificador IDD_SHEDULER el cual corresponde a la ventana *Algoritmos de Planificación*. Una vez que se ha seleccionado el identificador IDD_SHEDULER se presenta la ventana *Algoritmos de Planificación* en la ventana de edición del entorno de desarrollo de Visual C++, como se muestra en la Figura 5.3.3.

Una vez que aparece la ventana *Algoritmos de Planificación*, el programador elige un Botón de Opción de la ventana *Controls* que aparece en la ventana de edición como se muestra en la Figura 5.3.4, y se coloca en el grupo al cual pertenece el algoritmo de planificaón (Planificación por Prioridades Estáticas o Planificación por Prioridades Dinámicas). Posteriormente, se establecen la propiedades del nuevo botón efectuando un click izquierdo sobre el nuevo Botón de Opción para elegir la opción *Properties* (Propiedades). Esta opción permitirá establecer un identificador y proporcionar una etiqueta para identificar el nuevo algoritmo en la ventana *Algoritmos de Planficación* (como se muestra en la Figura 5.4).

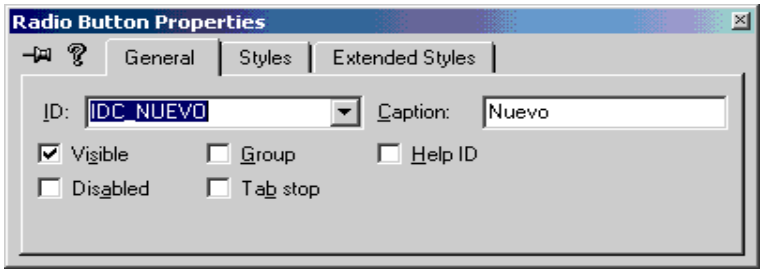

Figura 5.4: Propiedades del Botón de Opción para el nuevo algoritmo que se añade a PlataTR

Para finalizar con el proceso de añadir un nuevo algoritmo, se hace un nuevo click en el

botón *Aplicar* de la ventana de *Algoritmos de Planificación* de PlataTR para escribir código dentro de una estructura IF-ELSE la cual permitirá asignar un número al nuevo algoritmo (hasta el momento se tiene asignados los números del 0-5) que servirá para identificarlo en la clase CReal.

5.2.5 Clase CReal

La clase *CReal* es el núcleo de PlataTR, ya que aquí es donde se han implementado las operaciones relacionadas con la planificación de tareas de tiempo real. En esta clase se encuentra declarada la estructura CArrayTask la cual permite construir el arreglo de tareas de la cola de listos de PlataTR. La cola de listos contiene a las tareas listas para ejecución.

En esta clase, se implementá la estructura denominada CTrazaRect, la cual permite almacenar las coordenadas gráficas de cada tarea que ha sido planificada en cada tick generado por la función SetTimer de Windows. La clase CReal es una instancia de las clases CGraphics y CPlataView.

Las funciones implementadas en esta clase son:

- **Scheduling()**. Esta función es la encargada de dar un orden a la ejecución de las tareas. La próxima tarea a ejecutarse se selecciona de acuerdo a los parámetros temporales de cada tarea y de acuerdo a la política de planificación utilizada para la simulación (RM, DM, EDF, LLF).

  En la subsección 5.2.4 se mostro como añadir un nuevo algoritmo de planificación en la interfaz gráfica de PlataTR. Sin embargo, no se explicó como vincular al algoritmo con su respectivo código ejecutable. Es precisamente en la función Scheduling() donde se asocia el código respectivo al nuevo algoritmo de planificación. La Tabla 5.1 muestra parte del código donde se define la política de planificación del nuevo algoritmo. Las líneas 40 a la 47 de la Tabla 5.1 muestran el modelo para insertar un algoritmo de planificación.

- **MissedDeadline()**. Esta función comprueba en cada instante de tiempo (tick) si alguna tarea ha perdido su plazo de respuesta. Si ha ocurrido la pérdida de plazo de respuesta de una tarea, se actualizan los parámetros relativos de la tarea (m_ADeadlineRel

```
1. int CReal::Scheduling()
2. {
3. if (task2.NumTask != 0 && task1.EndJob == 0)
4.   switch(Policy)
5.   {
6. case 0:     // RM
7. AssignPrioRM();
8. {
9.    if (task1.PeriodoAbs <= task2.PeriodoAbs)
10.       task1 = task1;
11.    else
12.       task1 = task2;
13.    break;
14. }
15. case 1:     // DM
16. AssignPrioDM();
17.  {
18.   if (task1.DeadlineAbs <= task2.DeadlineAbs)
19.       task1 = task1;
20.    else
21.       task1= task2;
22.    break;
23.   }
24. case 2:     // EDF
25. {
26.    if (task1.DeadlineAbs <= task2.DeadlineAbs)
27.       task1 = task1;
28.    else
29.       task1 = task2;
30.    break;
31. }
32. case 3:   // LLF
33. {
34.   if((task1.DeadlineRel-time-task1.Exec)<=(task2.DeadlineRel-time-task2.Exec))
35.       task1 = task1;
36.   else
37.       task1 = task2;
38.     break;
39. }
40. case N:   // NUEVO ALGORITMO
41. {
42.    Si se cumple con la condición (política de planificación)
43.       task1 = task1;
44.    de lo contrario
45.       task1 = task2;
46.     break;
47.  }
48. }
49.}
```

Tabla 5.1: Función utilizada para introducir un nuevo algoritmo de planificación

y m_APeriodoRel) y el tiempo de cómputo se reinicia en cero.

- **CheckPeriodo()**. Esta función verifica si se termina el período de las tareas y calcula el nuevo período relativo, reiniciando las variables (tiempo ejecutado = 0) para la tarea que ha iniciado un nuevo período .

- **DrawRealTask()**. Recibe la tarea seleccionada por la función Scheduling() (tarea planificada), e incia con la etapa de dibujo en la pantalla.

- **DrawTickLine()**. Esta función controla el movimiento de la línea de dibujo cada vez que se genera un tick nuevo.

- **DrawCurrentTask()**. Esta función calcula las coordenadas para dibujar la gráfica de Gantt de la tarea que ha sido planificada.

5.2.6 Clase CPlataSlider

La Clase *CPlataSlider* permite controlar los botones de desplazamiento que se muestran en la Figure 5.2.3 de la ventana principal de la herramienta PlataTR. CPlataSlider permite verificar la posición de los botones de desplazamiento en el área de dibujo.

5.2.7 Clase CMisDatos

La clase *CMisDatos* implementa la ventana de diálogo *Parámetros de Entrada*. Esta clase permite introducir los parámetros temporales de un nuevo conjunto de tareas, mediante cajas de edición.

5.2.8 Clase CGridCtrlDlg

En la clase *CGridCtrlDlg* se implementa la ventana *Conjunto de Tareas* la cual permite visualizar los parámetros de las tareas mediante filas (donde cada fila representa una tarea del sistema) y columnas (cada columna representa un parámetro especifico). Además, proporciona botones para Insertar o Eliminar tareas del sistema, y se implementa un botón para cambiar el tipo y tamaño de la letra con la que se visualiza los textos en la ventana *Conjunto de Tareas*.

5.2.9 Clase CGridCtrl

La clase *CGridCtrl* se encarga de controlar la rejilla que aparece en la ventana *Conjunto de Tareas*. Esta clase controla el número de columas y filas de la rejilla que deben ser mostrados dependiendo del número de parámetros y número de tareas. Además, controla la anchura y altura de las filas y las columnas.

5.2.10 Clase CGridCellBase

La clase *CGridCellBase* permite controlar el color de cada una de las celdas de la rejilla. Al seleccionarse una celda en particular, ésta tomará un color diferente a las demás celdas. Con esta clase también se controla el evento que se generá (y se habilita la celda para editar un parámetro) al hacer doble click en alguna celda.

5.3 Ejemplos

En esta sección presentamos algunos ejemplos de planificación de tareas periódicas de tiempo real utlizando PlataTR. Primero mostraremos como se obtiene la cacterización y el modelado de una aplicación de tiempo real, se muestra el modelado de un conjunto de tareas generado por un Micro-Kernel de tiempo real y por último se muestran algunos ejemplos en los que se conoce los parámetros temporales de cada tarea.

5.3.1 Ejemplo de una Aplicación de Tiempo Real

En esta sección se detallará una aplicación de un sistema de tiempo real, su caracterización y su modelado en la herramienta gráfica PlataTR (modo estático).

Los pasos que se siguen en el diseño de un sistema en tiempo real son los siguientes:

Paso 1. Identificación del sistema:

En este paso se describe la operación del sistema y se identifican los componentes del sistema.

El sistema que se muestra en la Figura 5.5, permite regular el nivel y la temperatura del líquido que llega al tanque. Cuando la cantidad de líquido en el tanque sea demasiado escasa o cuando la cantidad de líquido este cerca de rebasar el nivel máximo permitido, será preciso

abrir o cerrar la válvula de escape. Por otro lado, cuando la temperatura sea más alta que la permitida o sea demasiado baja, se controlará la cantidad de calor que alimenta al tanque por medio de un calefactor.

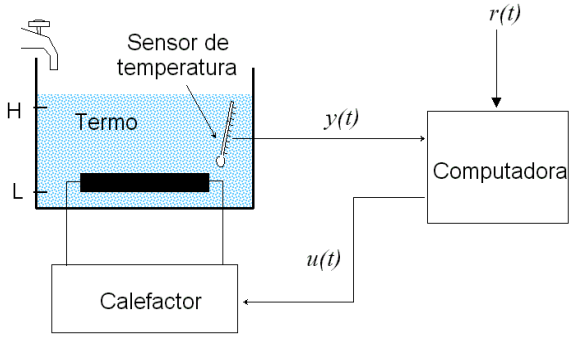

Figura 5.5: Sistema de tiempo real que regula temperatura y nivel de líquido en un Tanque.

La Figura 5.6 muestra paso a paso como se debe realizar el proceso de planificación de una aplicación de tiempo real el cual es aplicado para el análisis en PlataTR.

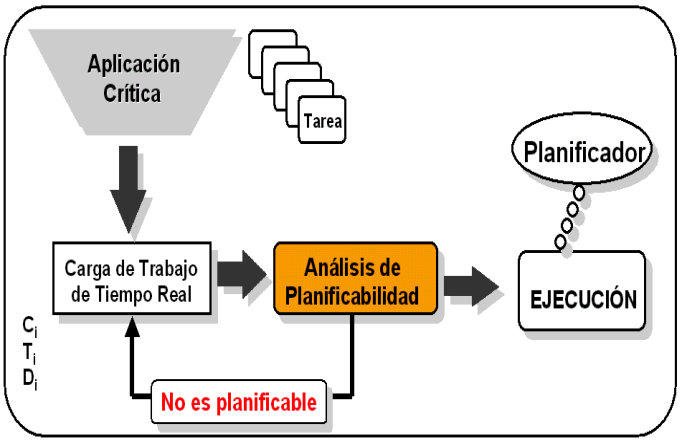

Figura 5.6: Proceso de planificación

Los componentes del sistema son: tanque, sensores, actuadores, interfaces A/D, y computadora.

Una vez que se han identificado las funciones y componentes del sistema, la siguiente actividad consiste en planificar el sistema. La Figura 5.6 muestra el proceso de planificación, en el que se identifican los pasos: caracterización, análisis de planificabilidad, y ejecución del

sistema.

Paso 2. Caracterización del sistema:

En este paso, se identifican las tareas del sistema y sus restricciones temporales. Las tareas que permiten controlar el sistema de nuestro ejemplo son las siguientes:

1. Tarea 1: Control del nivel del líquido en el tanque.

2. Tarea 2: Control de la temperatura del líquido almacenada en el tanque.

Los parámetros temporales de la aplicación de tiempo real son básicamente, cómputo de las tareas, período y plazo de respuesta. El cómputo es un parámetro que se obtiene de medir el tiempo de cómputo de cada tarea (cuando se ejecuta de forma ininterrumpida), para una sola instancia de ejecución (período). Esta medición se puede llevar a cabo, utilizando el reloj de tiempo real del sistema, o utilizando analizadores lógicos o osciloscopios digitales. El período y el plazo se obtienen mediante la observación del sistema.

El período de la tarea 1 se podría obtener de la siguiente forma:

El nivel de líquido del tanque sube conforme el tanque se va llenando de agua. Por esto es necesario medir la velocidad con que el tanque esta siendo llenado. Una vez medida esta velocidad, podremos saber cuanto tiempo se tardaría el tanque en ser llenado, a partir de un estado vacío. Si el nivel de líquido esta subiendo a razón de 1 cm por segundo, y el tanque tiene 50 cms, podriamos verificar el nivel cada segundo. Este criterio permitiría controlar el líquido del tanque, a partir de cualquier nivel en que se encuentre.

El control del líquido del tanque podríamos programarlo mediante un controlador PID, en el cual el loop de control podría ser de 1 segundo. El controlador se encargaría de controlar el flujo de líquido a más lento o mas rápido, dependiendo del nivel en que se encuentre el líquido. En el caso mas extremo, el nivel de líquido estaría en su nivel máximo, y entonces la valvula del agua se cerraría por completo.

El plazo de respuesta de esta tarea estaría en función de la rapidez con la que esperamos se realize el cómputo de la tarea. En este caso, al plazo de respuesta le daremos el mismo valor que el período.

Los parámetros temporales de la tarea 2 se obtienen de la siguiente manera:

Para controlar la temperatura del sistema de tiempo real se tiene una señal $u(t)$ la cual indica la cantidad de calor que el calefactor debe suministrar, y una señal $y(t)$ la cual se obtiene de los sensores de la temperatura del líquido del tanque. Para mantener el líquido a una temperatura deseada se tiene una señal de referencia $r(t)$. Es decir, para que la temperatura se encuentre controlada $y(t)$ debe ser lo más perecida a $r(t)$.

Al igual que en la tarea 1, el tiempo de cómputo se obtiene de medir la ejecución de la tarea durante una instancia. Los cambios de temperatura son más drásticos cuando la temperatura suministrada es la mayor posible ($max\ u(t)$). La razón a la que la temperatura cambia estando suministrando calor a la mayor temperatura, es el período que podríamsos escoger para esta tarea. Al igual que en la tarea 1, en esta tarea el plazo sera igual al período.

La Figura 5.7 muestra los parámetros temporales que se han asignado a cada tarea.

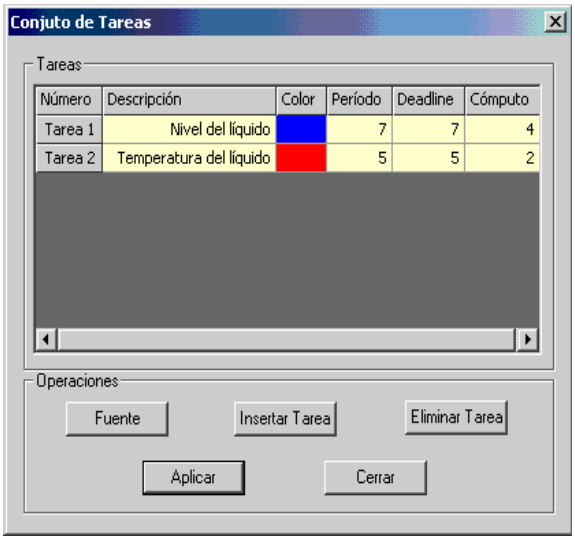

Figura 5.7: Primera asignación de parámetros temporales

Paso 3. Análisis de Planificabilidad (Ejecución en PlataTR).

En este paso, se verifica si los parámetros temporales escogidos permiten planificar el sistema, sin que las tareas pierdan sus plazos de respuesta. Una vez que se han asignado los parámetros temporales se elige el algoritmo de planificación.

En nuestro caso, el análisis de planificabilidad lo realizaremos mediante la simulación del sistema. En esta simulación, la herramienta PlataTR nos muestra gráficamente la ejecución

de las tareas del sistema y nos indica si el conjunto de tareas es o no planificable.

La Figura 5.8 muestra el comportamiento temporal del conjunto de tareas de la Figura 5.7, aplicando el algortimo RM (*Rate Monotonic*) en el intervalo de tiempo [0,30]. Como puede apreciarse la Tarea 1 pierde su plazo de respuesta en el instante de tiempo 7. Esto se debe a que sólo se ejecuta tres unidades de tiempo en su primer instancia. RM asignó la prioridad más alta a la tarea 2 (por contar con el período más pequeño).

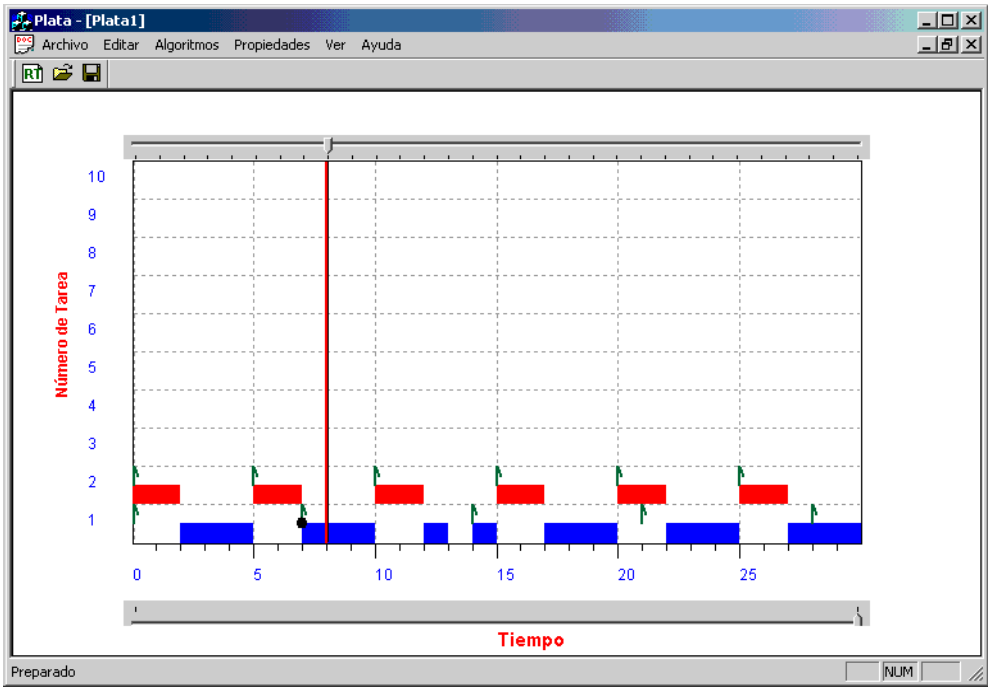

Figura 5.8: Planificación con RM falla.

El diseñador de sistemas de tiempo real debe buscar una solución que garantice que el conjunto sea planificable. Puede hacerlo de dos formas:

1. **Modificando los parámetros temporales.**

   La Figura 5.9 muestra la modificación de los parámetros temporales de las tareas que se mostraron en la Figura 5.7. Como se puede observar se modificó el plazo de respuesta de la tarea 1, ahora es $D_i = 8$. La Figura 5.10 muestra el comportamiento de las tareas utilizando el algoritmo RM, en el intervalo [0,30], como puede apreciarse no existe pérdida de plazo.

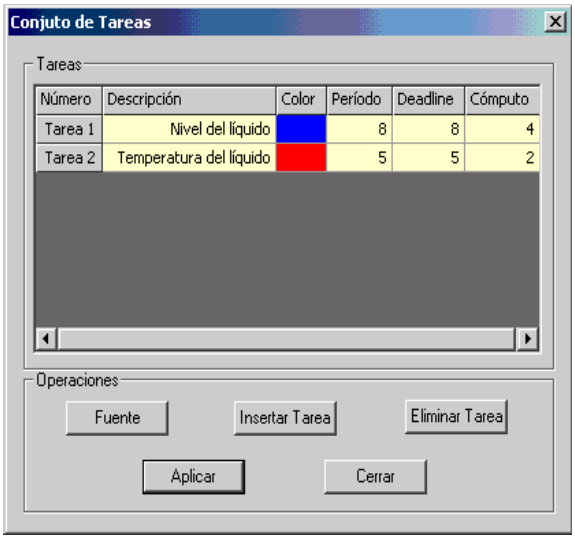

Figura 5.9: Primera asignación de parámetros temporales

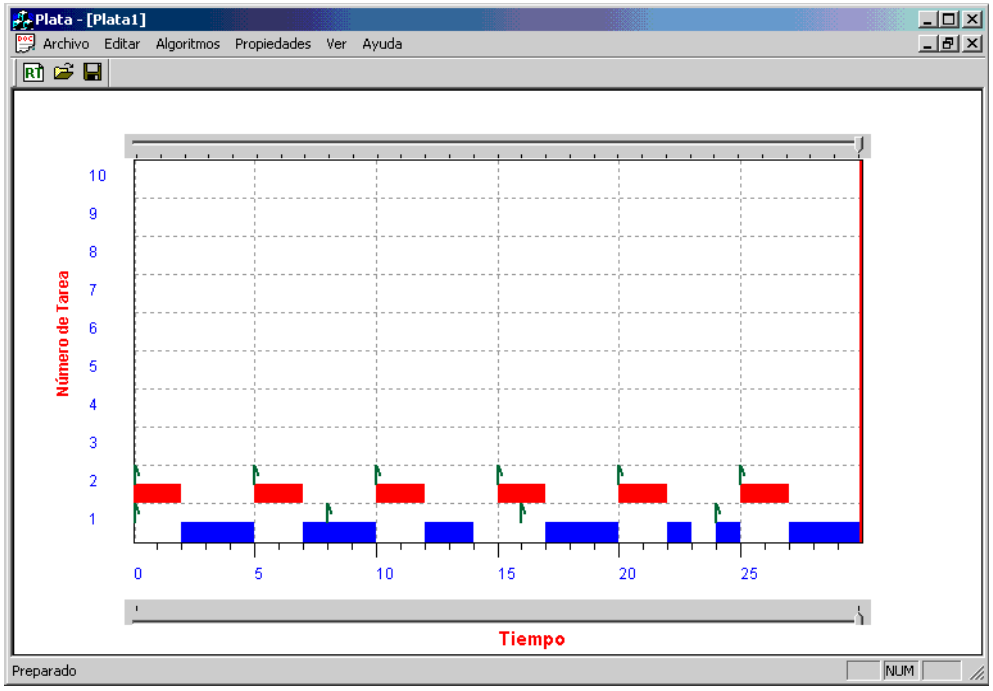

Figura 5.10: Planificación con RM, el conjunto es planificable.

2. **Probando con otro algoritmo de planificación.**

   La segunda forma de buscar que el conjunto de tareas sea planificable, es verificar el comportamiento del conjunto de tarea aplicando otro algoritmo de planificación. Para

el caso de nuestro ejemplo, utilizaremos los primeros parámetros temporales asignados a cada tarea, como se muestran en la Figura 5.7, pero ahora aplicando el algoritmo de planificación EDF.

La Figura 5.11 muestra en comportamiento temporal en el intervalo de tiempo [0,30]. Como puede apreciarse el conjunto de tareas es planificable bajo EDF.

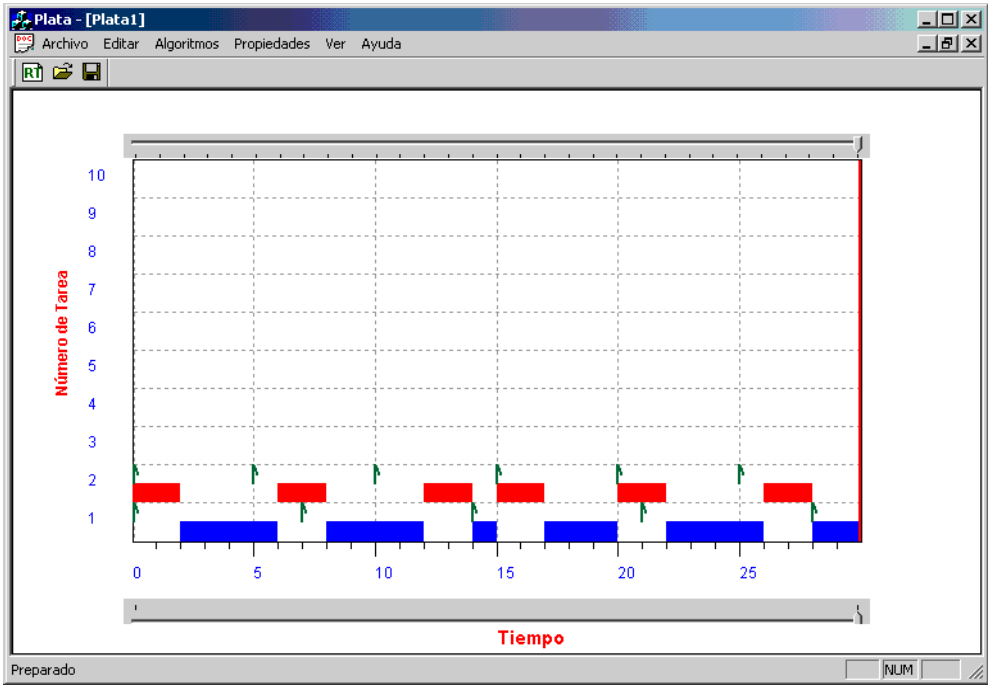

Figura 5.11: Planificación con EDF, con los parámetros asignados en la Figura 5.7.

5.3.2 Ejemplo de una Aplicación generada por el Micro-Kernel

En esta sección se detallará un conjunto de tareas generado por el Micro-Kernel de tiempo real, su caracterización y su modelado en la herramienta gráfica PlataTR (modo dinámico).

Los pasos que se siguen para el modelado y graficación de un sistema de tiempo real generado por el Micro-Kernel de tiempo real son los siguientes:

Paso 1. Caracterización del sistema:

En este paso, se identifican las tareas del sistema y sus restricciones temporales. Este ejemplo permitirá visualizar un conjunto de tareas previamente ejecutado por el Micro-Kernel de tiempo real desarrollado en la Sección de Computacón. PlataTR contiene un módulo que permite ejecutar las tareas generadas por el Micro-Kernel de Tiempo Real.

La tabla 5.2 muestra los parámetros temporales de cada tarea, y los eventos generados por el Micro-Kernel de tiempo real. La primera fila de la tabla 5.2 propociona a PlataTR el número de tareas que genero el Micro-Kernel de tiempo real. Las siguientes filas proporcionan los parámetros temporales: período y tiempo de cómputo.

Durante la ejecución del Micro-Kernel de tiempo real, se generan distintos eventos que caracterizan a la ejecución del conjunto de tareas, y que serán utilizados por PlataTR.

Los distintos eventos son:

- **I**. I respresenta el inicio de ejecución de una tarea. Es decir, el planificador del Micro-Kernel de tiempo real asignó el procesador a una tarea.
- **F**. F representa el instante en que la tarea fue desalojada del procesador o la tarea a terminado su ejecución por el planificador del Micro-Kernel de tiempo real.
- **PP**. PP representa la pérdida de plazo de una tarea.

En la tabla 5.3 se muestran los eventos generados durante la planificación del conjunto de tareas generada por el Micro-Kernel. A cada evento se le ha asignado durante la graficación, una etiqueta para su identificación. Esta tabla en realidad es un archivo, que PlataTR lee para graficar la ejecución de las tareas.

La Figura 5.12 muestra los parámetros temporales de la tareas leidas por PlataTR. Como se puede observar, PlataTR asigna un color a cada tarea con la finalidad de poder identi-

4		
Tarea	T_i	C_i
1	45	15
2	120	30
3	160	10
4	300	25

Tabla 5.2: Tareas generadas por el Micro-Kernel de Tiempo Real

Evento	Tarea	Tiempo t	Evento	Tarea	Tiempo t	Evento	Tarea	Tiempo t
I	1	0	F	4	120	I	3	200
F	1	15	I	2	120	F	3	225
I	2	15	F	2	135	I	1	225
F	2	45	I	1	135	F	1	240
I	1	45	F	1	150	I	3	240
F	1	60	I	2	150	F	3	255
I	3	60	F	2	165	I	2	255
F	3	90	I	4	165	F	2	270
I	1	90	F	4	180	I	1	270
F	1	105	I	1	180	F	1	285
I	3	105	F	1	195	I	2	285
F	3	115	I	4	195	F	2	300
I	4	115	F	4	200	I	4	300

Tabla 5.3: Eventos generados por el Micro-Kernel de Tiempo Real

ficarla dentro del área de dibujo en el momento de la visualización. El color es asignado aleatoriamente por PlataTR.

Paso2. Análisis de Planificabilidad (Ejecución en PlataTR)

En este paso se gráfica el comportamiento temporal generado por el Micro-Kernel de tiempo real. Las Figuras 5.13, 5.14, 5.15 muestran gráficamente la secuencia temporal de las tareas generadas por el Micro-Kernel de tiempo real. La Figura 5.13 muestra la visualización en el intervalo de tiempo [0,30]. El archivo de eventos mostrado en la tabla 5.3 indica que

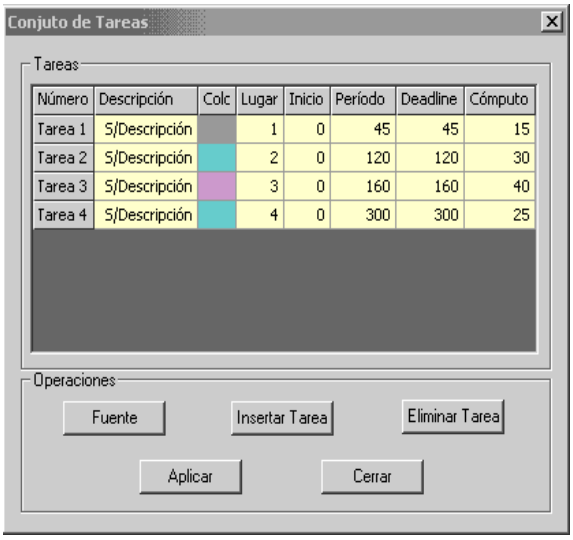

Figura 5.12: Muestra de las tareas que PlataTR ha recibido del Micro-Kernel de Tiempo Real

la primer tarea que fue planificada por el Micro-Kernel de tiempo real es la tarea 1 (Evento I, tarea 1, tiempo 0). Esta tarea se ejecuta desde el instante 0 al instante 15 (evento F, tarea 1, tiempo 15), y termina el tiempo de cómputo que le fue asignado. Siguiendo con la secuencia gráfica de la Figura 5.13 la siguiente tarea que fue seleccionada para su ejecución por el planificador del Micro-Kernel es la tarea 2 (evento I, tarea 2, tiempo 15).

La Figura 5.12 muestra la continuación de la ejecución de la tarea 2 la cual termina en el instante 45 (evento F, tarea 2, tiempo 45).

La Figura 5.14 muestra la visualización gráfica en el intervalo de tiempo [30,60]. Es posible observar que en el tiempo 45 la tarea 2 finaliza su ejecución (evento F, tarea 2, tiempo 45).

Continuando con la visualización gráfica, en la Figura 5.15 se muestra el intervalo de ejecución [89,119]. En el instante 90 la tarea 3 termina su ejecución (evento F, tarea 3, tiempo 90) y en ese mismo instante la tarea 1 inicia su ejecución (evento I, tarea 1, tiempo 90). De aquí la tarea 1 continua su ejecución hasta el instante 105 (evento F, tarea 1, tiempo 105). En este mismo instante de tiempo se inicia la ejecución de la ta tarea 3 (evento I, tarea 3, tiempo 105). Finalmente, en la Figura 5.15 se puede observar que en el instante de tiempo 115 se inicia la ejecución de la tarea 4 (evento I, tarea 4, tiempo 115), la cual finaliza su ejecucón en el instante de tiempo 120 (evento F, tarea 4, tiempo 120).

Como se pudo observar, PlataTR toma los eventos generados por el Micro-Kernel, me-

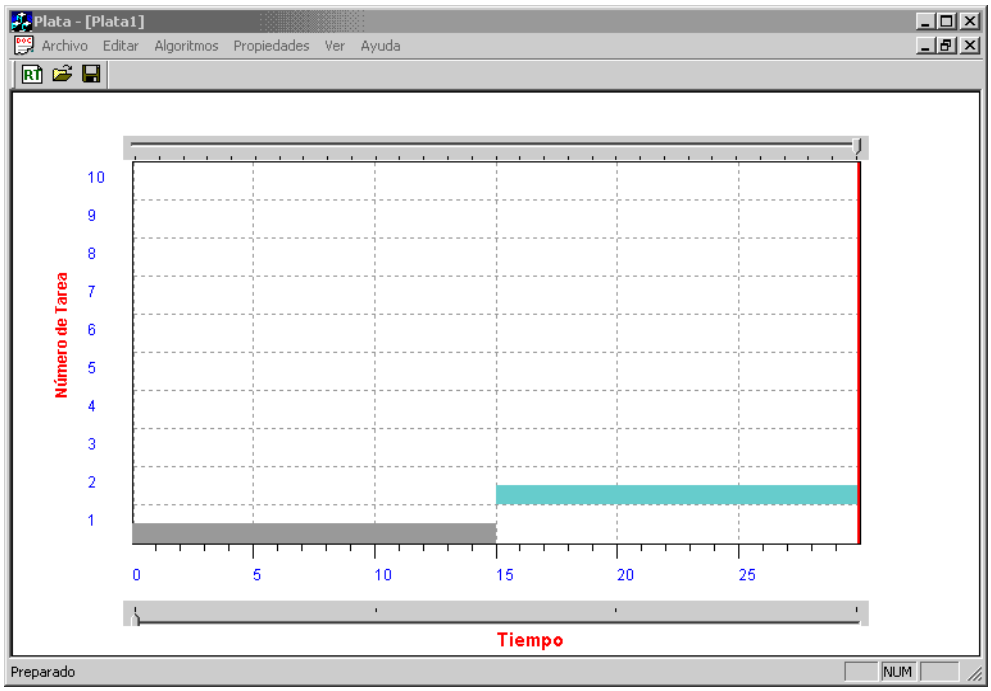

Figura 5.13: Visualización de los eventos en el intervalo de tiempo [0,30]

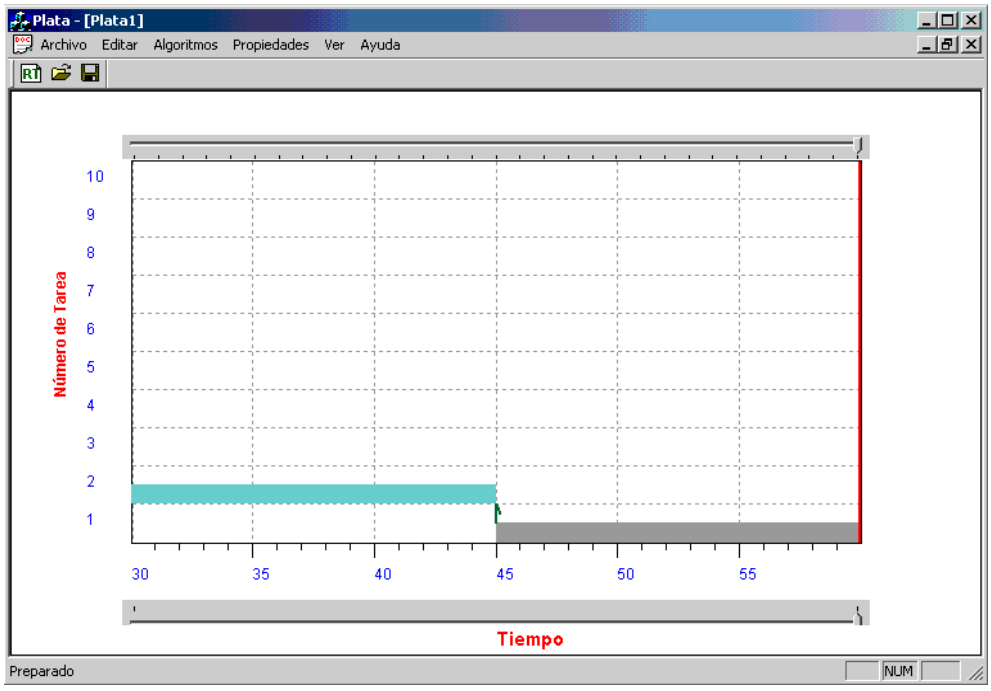

Figura 5.14: Visualización de los eventos en el intervalo de tiempo [30,60]

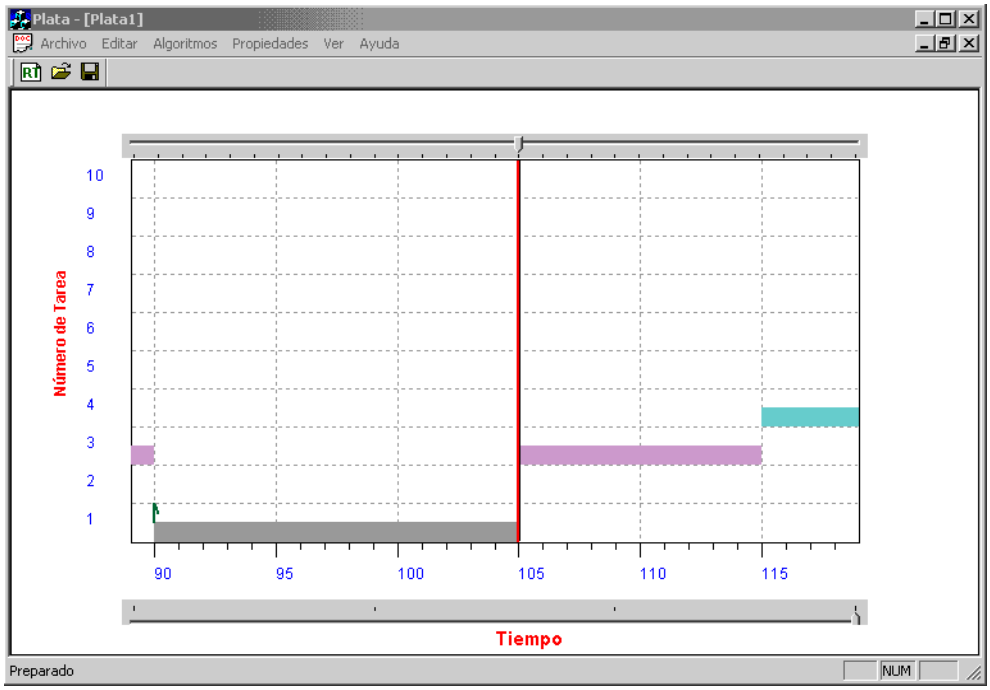

Figura 5.15: Visualización de los eventos en el intervalo de tiempo [89,119]

diante un archivo de eventos, para mostrarlos gráficamente.

5.3.3 Ejemplos con Parámetros Temporales Conocidos

En esta sub-sección se muestran varios ejemplos aplicando distintos algoritmos de planificación cuando se conocen los parámetros temporales del conjunto de tareas.

Ejemplo 1. La Figura 5.16 muestra un conjunto de tareas de tiempo real con sus parámetros temporales respectivos, los cuales fueron introducidos desde PlataTR para su análisis. Primero aplicamos el algoritmo de planificación Rate Monotonic. Debemos recordar que el algoritmo de planificación Rate Monotonic asigna la mayor prioridad a las tareas con menor período.

En la tabla 5.4 pueden observarse las prioridades que asigna a cada tarea el algoritmo de planificación Rate Monotonic.

La Figura 5.17 muestra la secuencia gráfica en el intervalo de tiempo [0,30] generada por PlataTR al aplicar el algoritmo de planificación Rate Monotonic. Como puede observarse, en el instante de tiempo 7 la tarea 4 solo se ejecuta una unidad de tiempo, ya que en ese

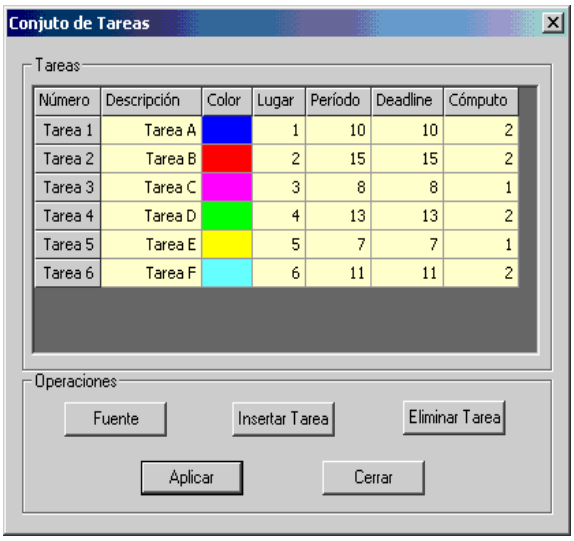

Figura 5.16: Especificaciones temporales ejemplo 1

Tarea	*Prioridad*
1	3
2	6
3	2
4	5
5	1
6	4

Tabla 5.4: Prioridad asignada a las tareas del ejemplo 1 por Rate Monotonic

instante es desalojada por la tarea 5. Esto se debe a que la tarea 5 tiene mayor prioridad que la tarea 4. Sin embargo la tarea 4 no se puede ejecutar cuando se termina de ejecutar la tarea 5 debido a que la tarea tres se activa en el instante 8, y esta tiene mayor prioridad que la tarea 4. A pesar de haber sido desalojada la tarea 4 por las tareas 5 y 3, esta termina su ejecución en el instante de tiempo 10, sin perder su plazo de respuesta. También podemos observar que la tarea 2 pierde su plazo de respuesta en el instante de tiempo 15. Esto se debe a que esta tarea tiene la menor prioridad.

Las Figuras 5.18 y 5.19 muestran la secuencia de planificación para los intervalos de ejecución [30,60] y [60,90]. Como puede observarse en estos dos intervalos de tiempo ninguna

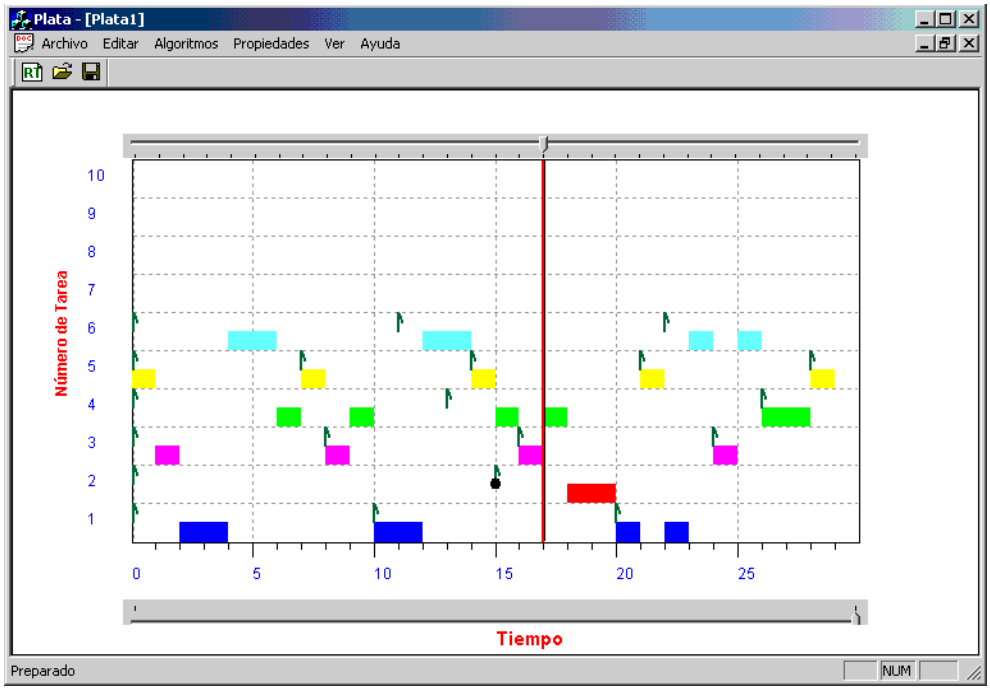

Figura 5.17: Secuencia temporal del ejemplo 1 bajo el algoritmo RM, intervalo [0,30]

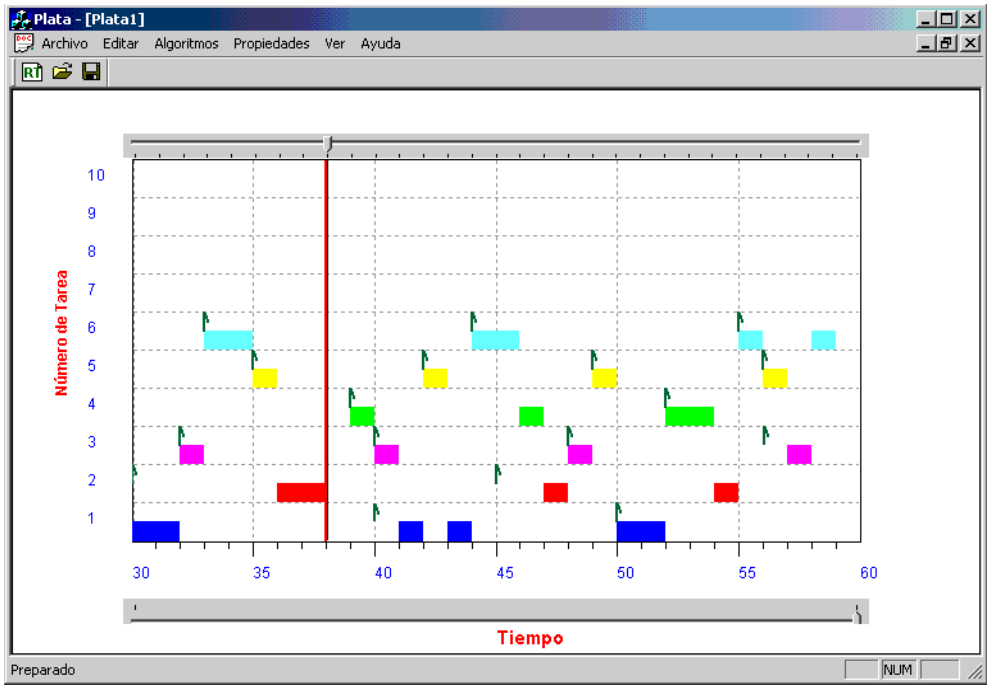

Figura 5.18: Ssecuencia temporal del ejemplo 1 bajo el algoritmo RM, intervalo [30,60]

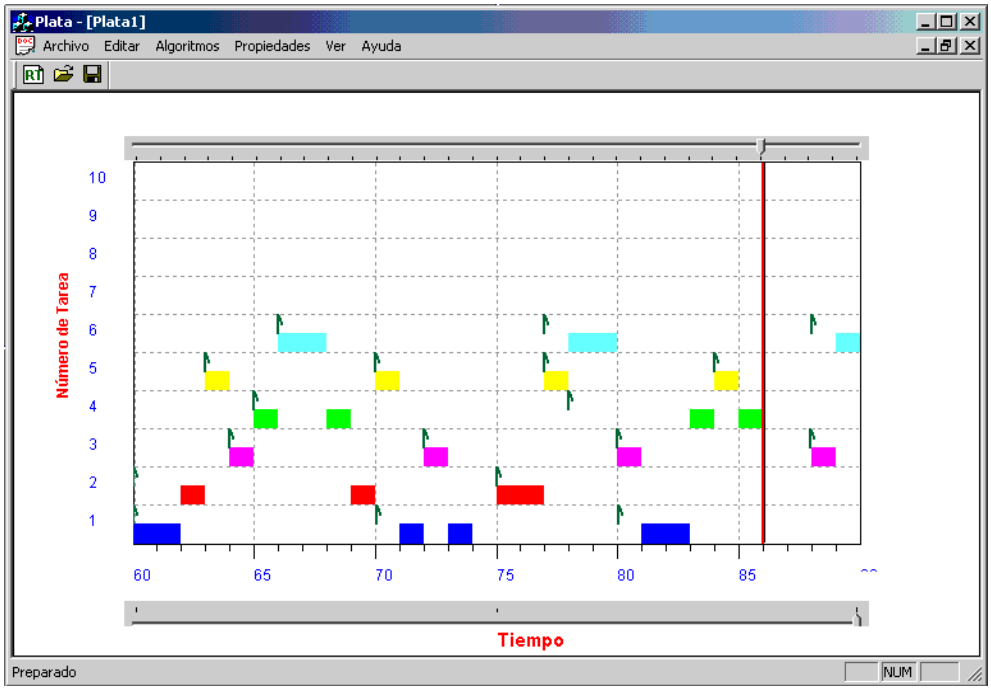

Figura 5.19: Secuencia temporal del ejemplo 1 bajo el algoritmo RM, intervalo [60,90]

tarea piede su plazo de respuesta.

Se utilizó también la política de planificación EDF con el conjunto de tareas definido en la Figura 5.16. Como podemos observar en las Figuras 5.20, 5.21 y 5.21, ninguna tarea pierde su plazo para el intervalo de tiempo [0,90]. Esto se debe a que EDF tiene una cota de utilización del 100%.

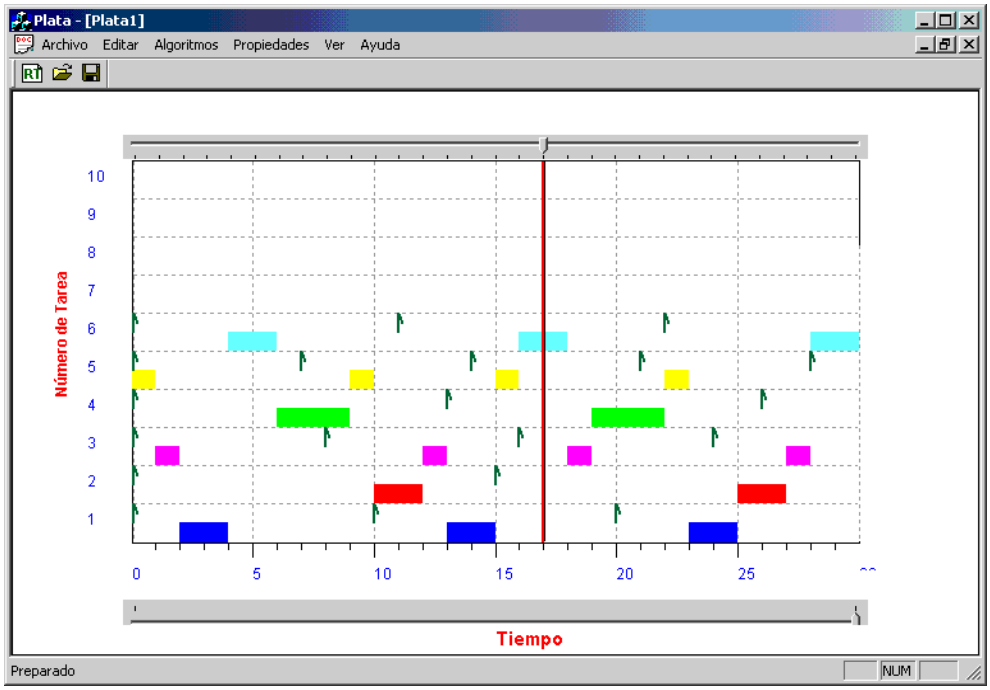

Figura 5.20: Secuencia temporal del ejemplo 1 bajo el algoritmo EDF, intervalo [0,30]

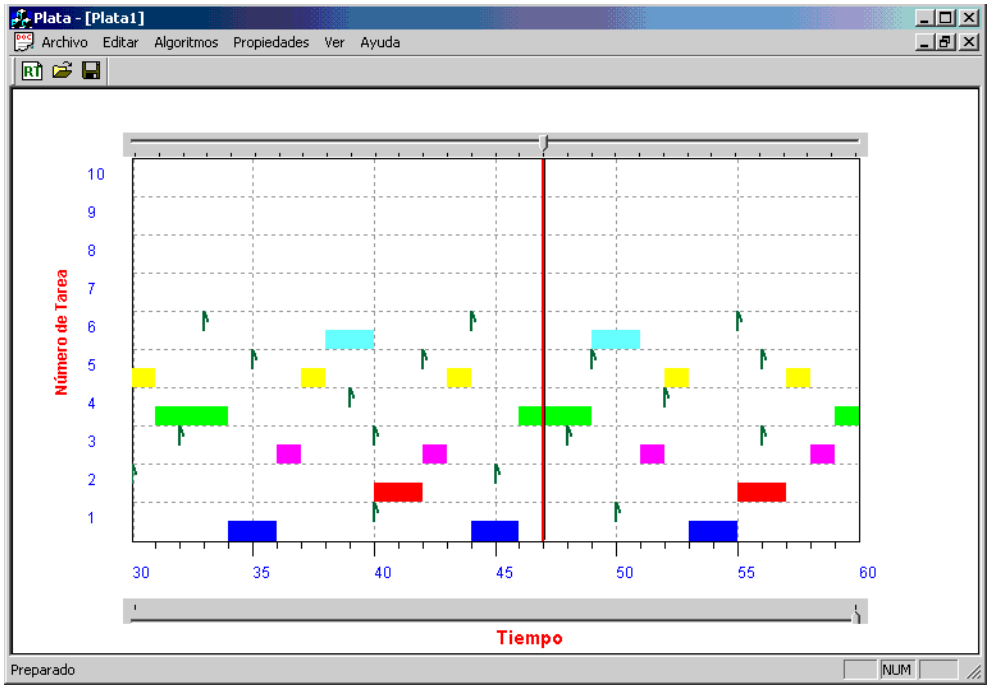

Figura 5.21: Secuencia temporal del ejemplo 1 bajo el algoritmo EDF, intervalo [30,60]

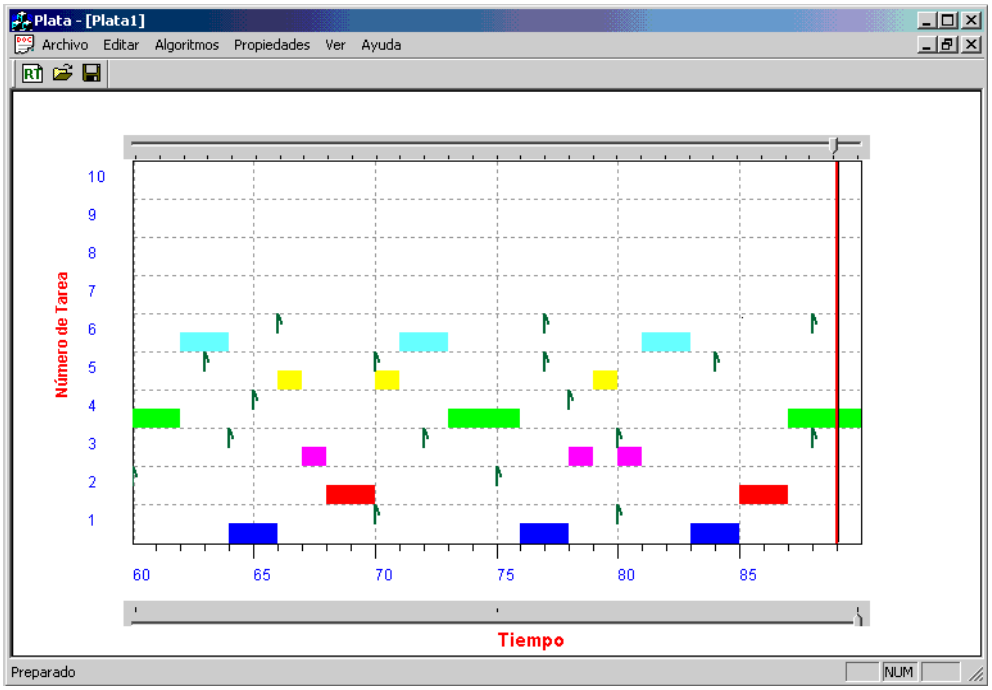

Figura 5.22: Secuencia temporal del ejemplo 1 bajo el algoritmo EDF, intervalo [60,90]

Ejemplo 2. La Figura 5.23 muestra un conjunto de tareas con sus respectivos parámetros temporales. Como puede observarse se trata de un conjunto de tareas armónico, lo cual indica que sus periódos son multiplos unos de otros. Ante esta condición, el conjunto de tareas tiene como cota de utilizacion el 100 %, como en el caso de EDF.

La ejecución de este conjunto de tareas, bajo el algoritmo RM la podemos observar en las Figuras 5.24 y 5.25. La Utilización total del conjunto de tareas mostrado en la Figura 5.23 es de $U_{TOT} = 0.9$ mientras que la cota límite de Liu y Layland, para un conjunto de 5 tareas es de $U_{min} = 5(2^{1/5} - 1) = 5(.1486) = 0.7334$. Por esto, se esperaría que el conjunto no sea planificable. Sin embargo, es planificable debido a que las tareas son armónicas.

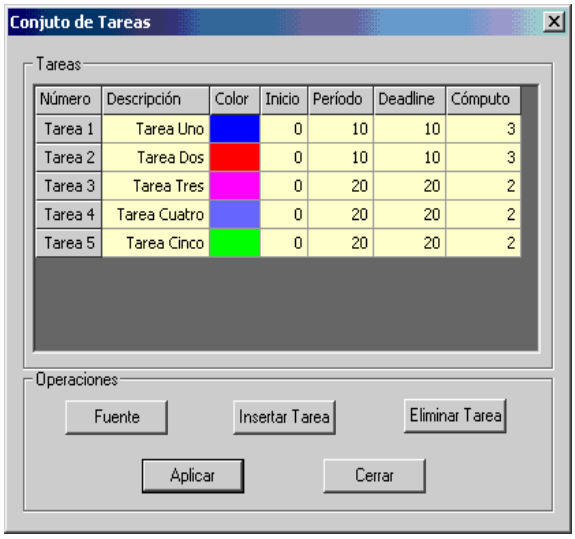

Figura 5.23: Especificaciones temporales para el ejemplo 2

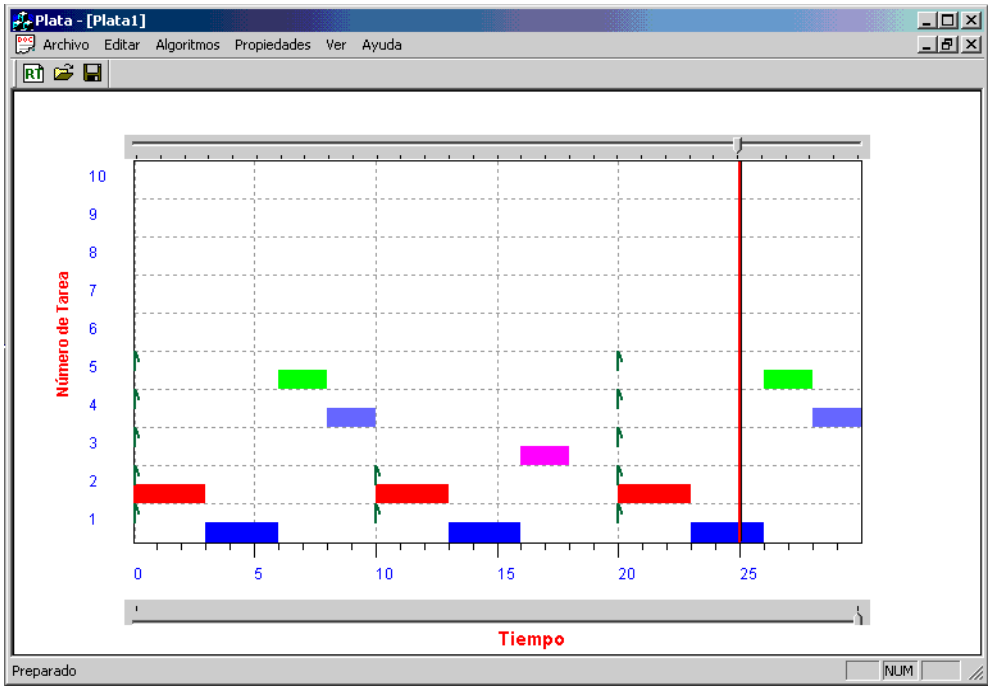

Figura 5.24: Secuencia temporal del ejemplo 2 bajo el algoritmo RM, intervalo [0,30]

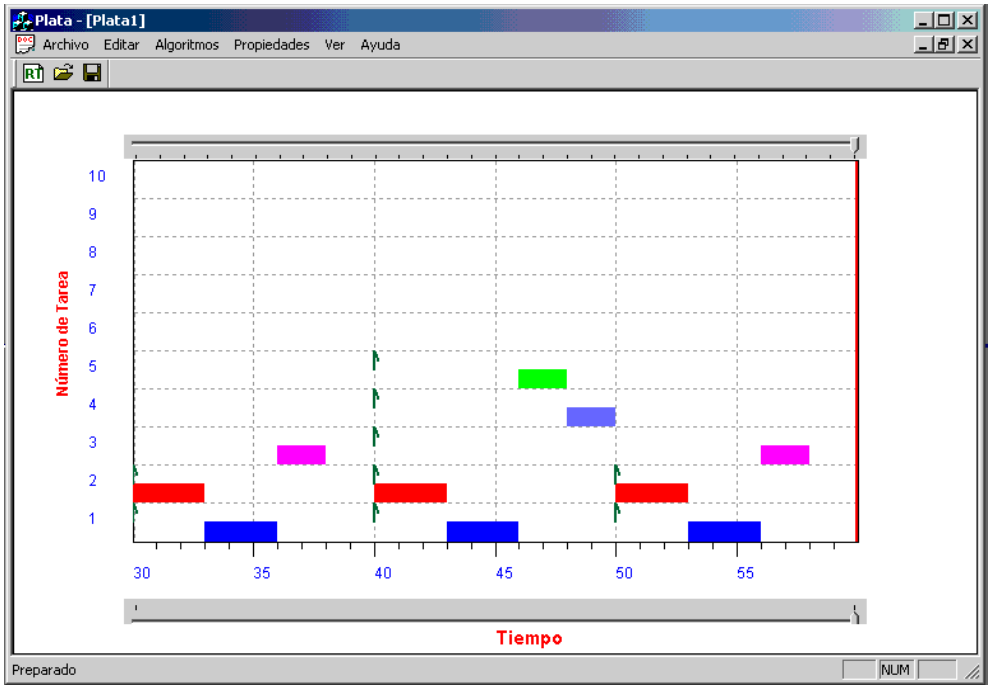

Figura 5.25: Secuencia temporal del ejemplo 2 bajo el algoritmo RM, intervalo [60,90]

Ejemplo 3. El siguiente conjunto de tareas que se muestra en la Figura 5.26 es un ejemplo de un conjunto de tareas que no es armónico. En este conjunto de tareas su utilización total es de $U_{TOT} = 0.8761$, la cual que es mayor a la cota límite para que sea planificado por RM (para tres tareas la cota límite es $U_{min} = n(2^{1/n} - 1) = 3(2^{1/3} - 1) = 3(0.2596) = 0.7788$. Al presentarse la condición $U_{TOT} > U_{min}$, no es posible saber (sin realizar un análisis de planificabilidad exacto) si el conjunto de tareas es o no planificable. Sin embargo, en la secuencia de ejecución presentada en las Figuras 5.27 y 5.28, podemos observar que ninguna tareas pierde sus plazos de respuesta durante el intervalo de tiempo [0,60].

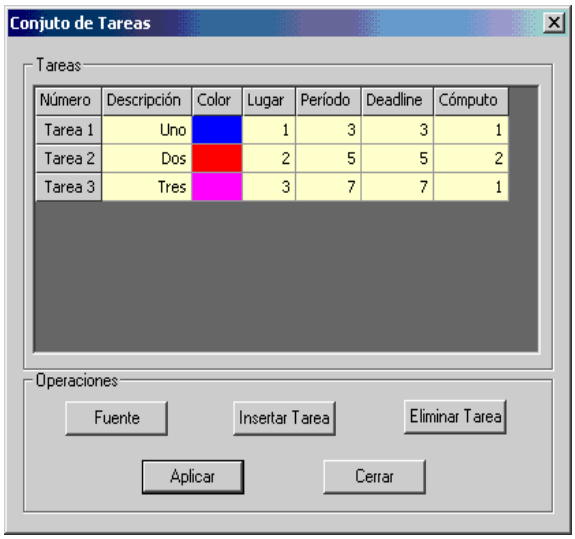

Figura 5.26: Especificaciones temporales para el ejemplo 3

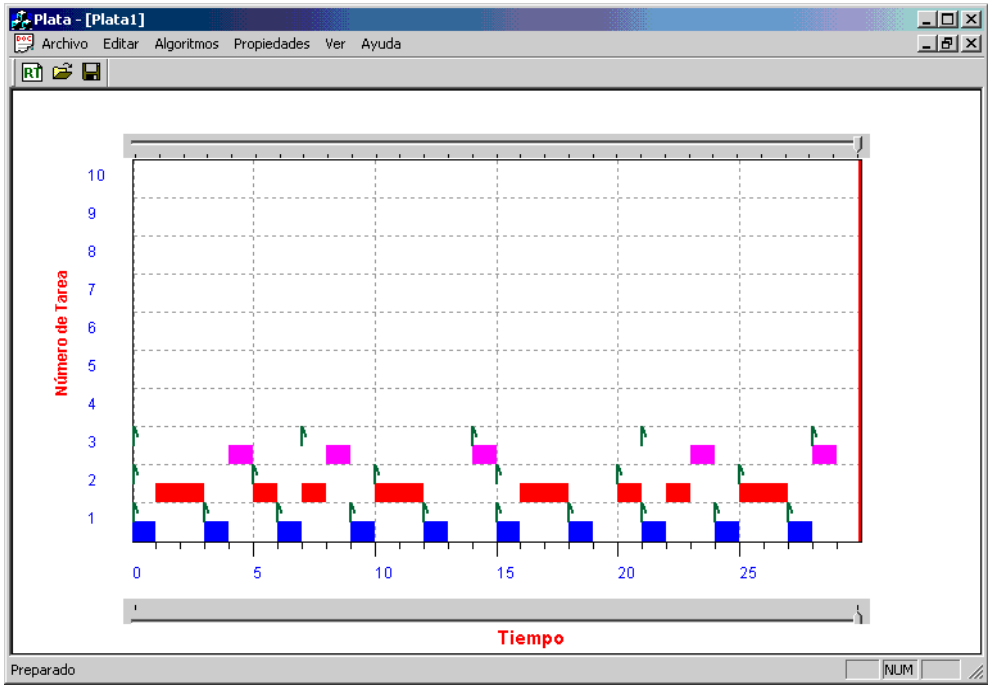

Figura 5.27: Secuencia temporal del ejemplo 3 bajo el algoritmo RM, intervalo [0,30]

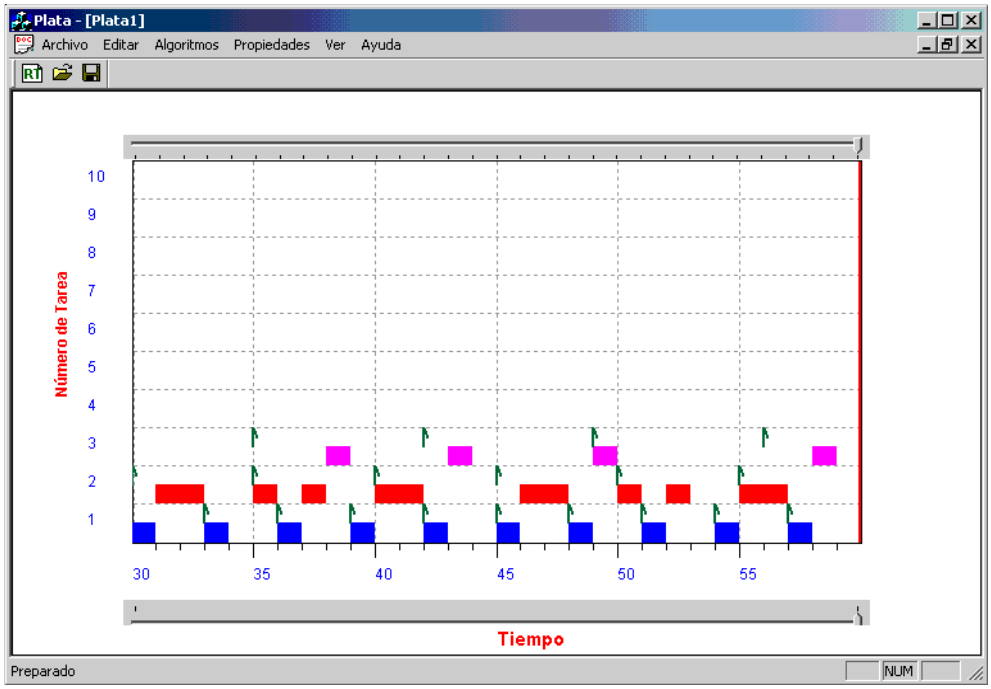

Figura 5.28: Secuencia temporal del ejemplo 3 bajo el algoritmo RM, intervalo [30,60]

Ejemplo 4. En este ejemplo se presentará la ejecución bajo PlataTR de un conjunto de tareas, utilizando la política de planificación EDF (Earliest Deadline First). La Figura 5.29 muestra las características temporales del conjunto de tareas a utilizar en la planificación. En el algoritmo EDF las prioridades se asignan de forma dinámica. Se asigna la prioridad más alta a la tarea con el plazo más cercano.

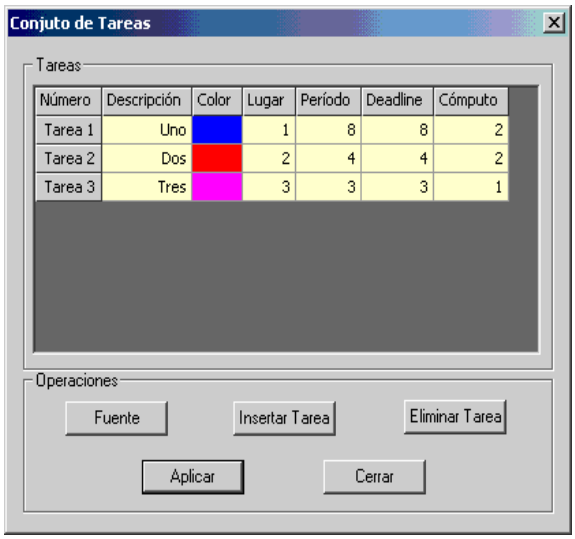

Figura 5.29: Especificaciones temporales para el ejemplo 4

La Figura 5.30 muestra la secuencia gráfica generada en PlataTR, como puede apreciarse la primera tarea que es ejecutada es *Tarea 3*, debido a que en el instante de tiempo $t = 0$ es la tarea que tiene su plazo más cercano $D_{Tarea3} = 3$. La tarea *Tarea 3* se ejecuta una unidad de tiempo que es el tiempo de cómputo que le fue asignado, de ahí que se actualiza el plazo de respuesta relativo el cual es $D_{Tarea3} = 6$. En el siguiente instante de tiempo $t = 1$, EDF asigna la prioridad más alta a la tarea *Tarea 2* debido a que es la tarea que tiene el plazo más cercano. Como puede observarse la prioridad más alta cambia en cada instante de tiempo (en base al plazo de respuesta más cercano). Otro aspecto que se puede apreciar en la Figura 5.30 es la pérdida de plazo de la tarea *Tarea 2* en el instante de tiempo 16, esto se debe a que EDF le asignó la menor prioridad en la cuarta instancia. Nuevamente, podemos apreciar otra pérdida de plazo de la tarea *Tarea 3* en el instante de tiempo 24, de hecho podemos observar que la tarea *Tarea 3* no alcanza a ejecutar en la octava intancia. La pérdidad de plazos se debe a que la utilización total del conjunto de tarea de la Figura 5.29 es $U = 1.08$

la cual es mayor a la cota de utilización establecida para garantizar la planificación de un conjunto de tarea ($U \leq 1$) bajo EDF.

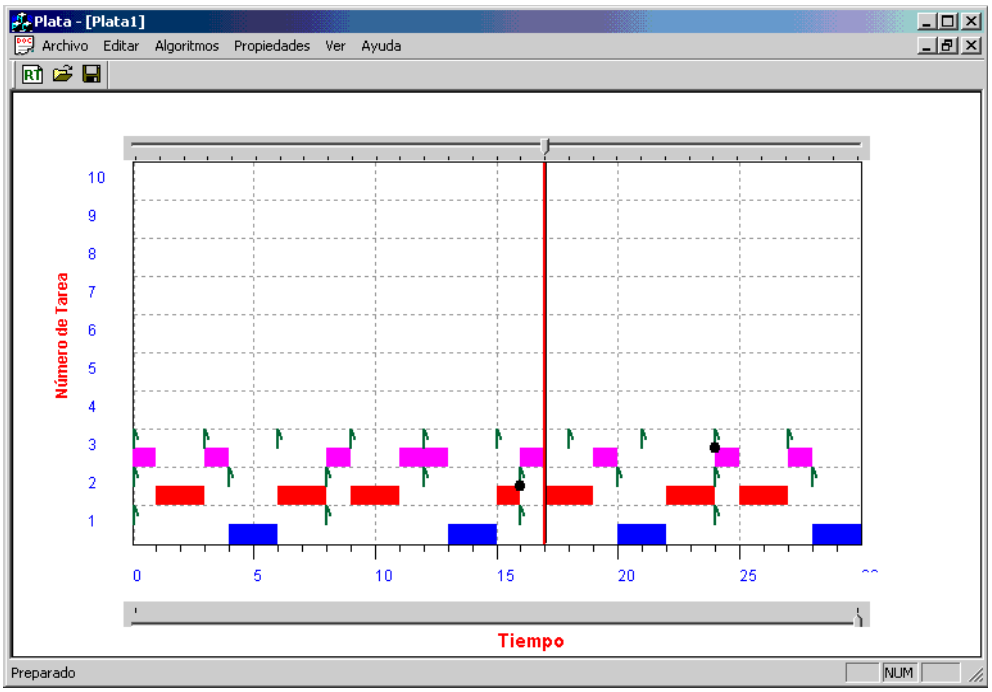

Figura 5.30: Secuencia gráfica ejemplo 4 bajo el algoritmo EDF, rango [0,30]

Ejemplo 5. El conjunto de tareas que se muestra en la Figura 5.31 será utilizado para mostrar la planificación mediante el algoritmo de planificación LLF (Least Laxity First). Debemos tener presente que LLF asigna prioridades a la tarea con menor holgura, donde la holgura se calcula de la siguiente forma, $holgura = D_i - tiempo - c_i$. En las Figuras 5.32, 5.33 y 5.34 se puede apreciar la planificación en el intervalo de tiempo [0,90]. En la Figura 5.32 puede apreciarse que la tarea 3, es la primer tarea seleccionada para ejecución por el planificador. Esto se debe a que tiene la menor holgura al tiempo 0. La Tabla 5.5 muestra el orden de ejecución para los primeros 6 instantes de tiempo, obtenido por LLF.

Tiempo	*Holgura*	*Tarea*
0	3	3
1	4	2
2	2	2
3	3	3
4	4	1
5	2	1
6	4	3

Tabla 5.5: Prioridad asignada en base a su holgura en cada instante de tiempo

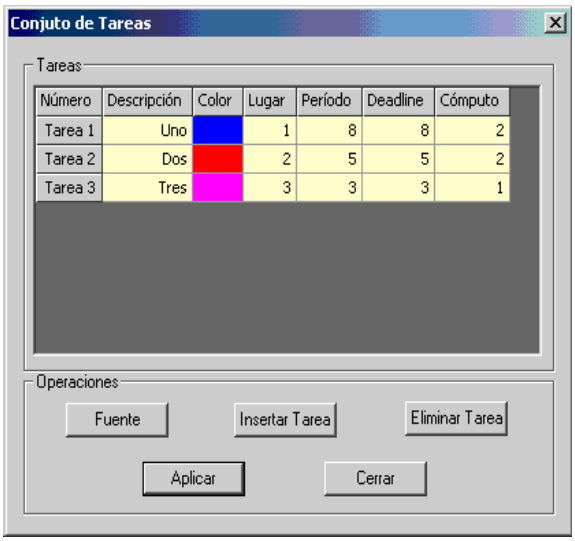

Figura 5.31: Especificaciones temporales para el ejemplo 5

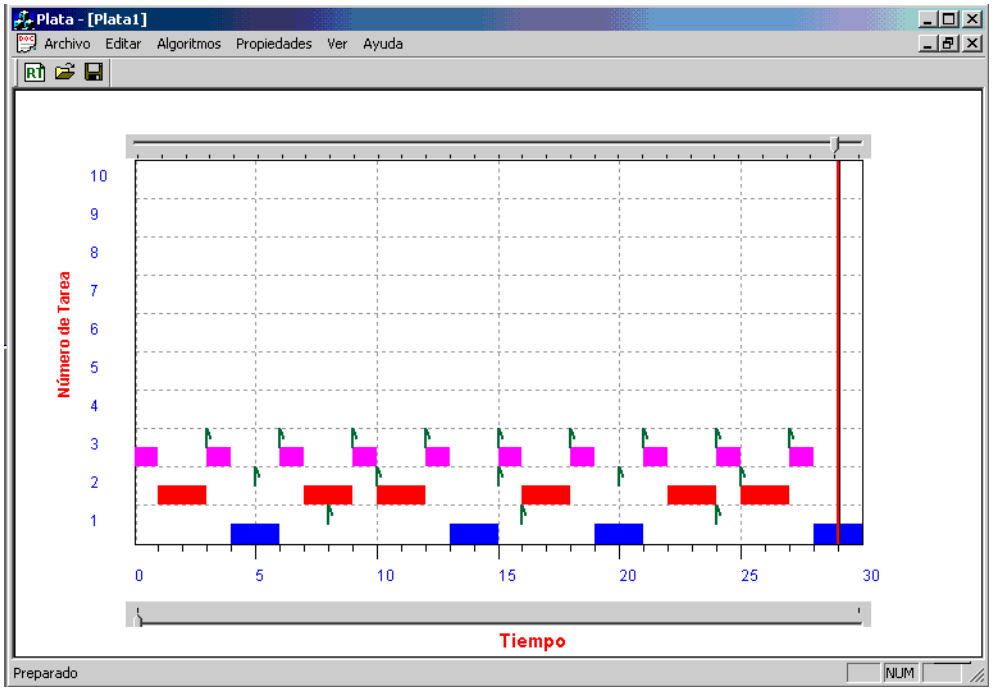

Figura 5.32: Secuencia gráfica del ejemplo 5 bajo el algoritmo LLF, para el intervalo [0,30]

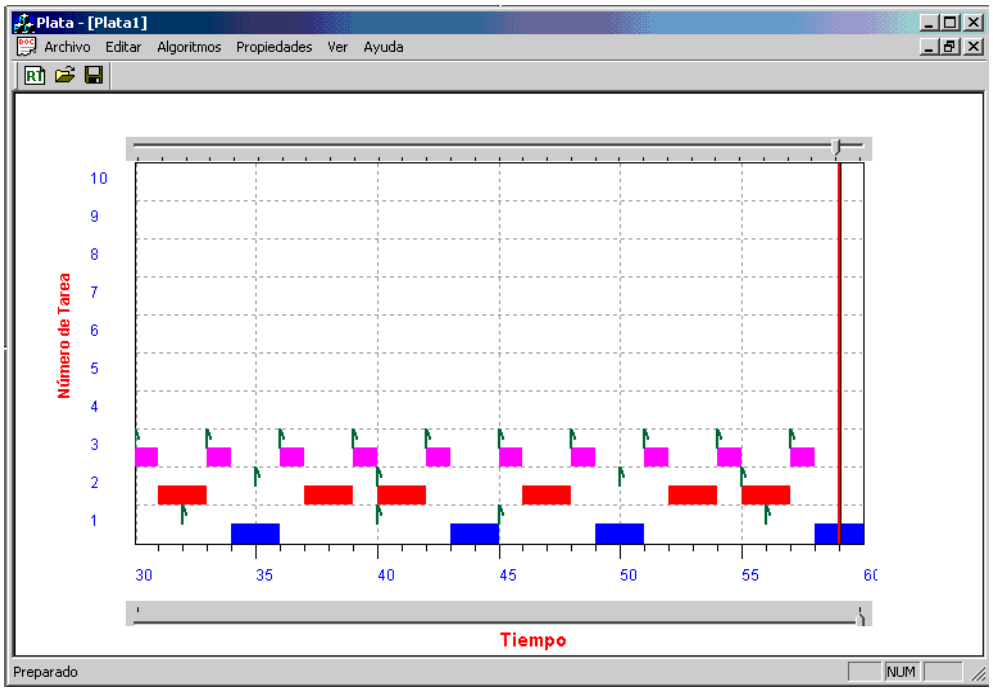

Figura 5.33: Secuencia gráfica ejemplo 5 bajo el algoritmo LLF, para el intervalo [30,60]

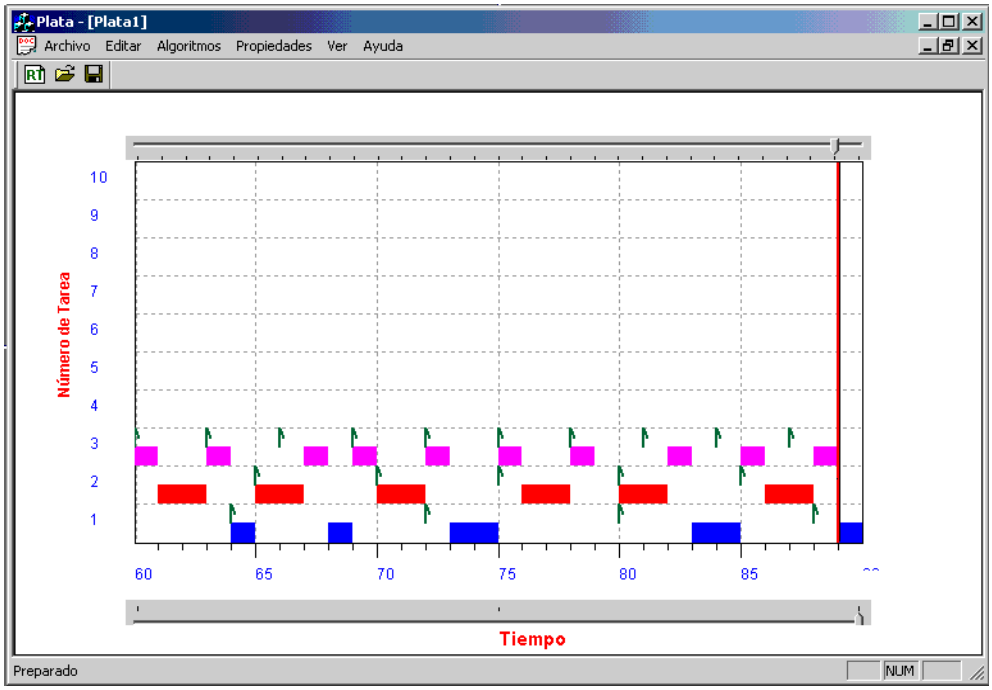

Figura 5.34: Secuencia gráfica ejemplo 5 bajo el algoritmo LLF, para el intervalo [60,90]

Capítulo 6

Conclusiones y Trabajo Futuro

6.1 Conclusiones

En este trabajo de tesis, se diseño una herramienta que permite verificar y validar de forma gráfica la planificación de un conjunto de tareas de tiempo real. Esta herramienta permite a un diseñador verificar el complimiento de restricciones temporales de un sistema de tiempo real. Si dichas restricciones no se cumplen, el diseñador puede cambiarlas en forma interactiva, o cambiar el algoritmo de planificación utilizado a fin de garantizar dichas restricciones.

En esta tesis nos enfocamos al estudio de algoritmos de planificación de sistemas de tiempo real para sistemas uniprocesadores. Se enumeraron los diferentes tipos de restricciones que pueden especificarse en las tareas de tiempo real, de las cuales la más importante para nuestro estudio son las restricciones de tiempo. Se describió también el problema de planificación, así como la clasificación de los algoritmos de planificación. Se describió el concepto de utilización, y se describieron los algoritmos de planificación más utilizados en los sistemas de tiempo real, el RM (Rate Monotonic), DM (Deadline Monotonic), EDF (Earliest Deadline First) y LLF (Laxity Least First), con un ejemplo donde se explica la planificación de estos algoritmos sobre un conjuto de tareas.

Se realizó un estudio de las herramientas gráficas de planificación de sistemas de tiempo real existentes en el mercado. Se constató que estas herramientas por lo general son muy complejas y son propietarias. Motivados por estos aspectos, en esta tesis decidimos realizar nuestra propia implementación.

Para lograr nuestro objetivo se implementó un modelo específico de tareas para simplificar el análisis de los algoritmo de planificación y se determinó la estructura lógica de la herramienta. Además, se presentaron las estructuras de datos que permitieron la implementación del modelo de tareas propuesto.

Finalmente, se implementaron ejemplos para verificar el funcionamiento de PlataTR, análizando varios conjuntos de tareas, aplicando los distintos algoritmos de planificación. Asi mismo, se mostro el comportamiento temporal de un conjunto de tareas generadas por un Micro-Kenel de tiempo real desarrollado en la Sección de Computación del Cinvestav-IPN.

Los resultados obtenidos de nuestro trabajo pueden resumirse en los cuatro puntos siguientes, que hacen referencia al cumplimiento de los objetivos generales y específicos de nuestro trabajo.

- El objetivo general consistió en el desarrollo de una herramienta gráfica la cual permite verificar, validar y monitorear un conjunto de tareas de tiempo real concurrentes utilizando una interface gráfica amigable.

- Se obtuvo un modelo de tareas acorde a los algoritmos de planificación implementados. Los algoritmos implementados fueron (*Rate Monotonic*, *Deadline Monotonic*), (*Earliest Deadline First*, y *Least Laxity First*).

- La herramienta PlataTR muestra gráficamente el comportamiento temporal dinámico de un conjunto de tareas generado por un Micro-Kernel de tiempo real.

- PlataTR se ejecuta bajo la plataforma de Microsoft Windows.

6.2 Trabajo Futuro

El trabajo a futuro consiste en continuar extendiendo la funcionalidad de la herramienta de simulación PlataTR. A continuación se presentan las líneas de trabajo de que se recomienda

realizar.

- Es necesario ampliar el modelo de tareas en los siguientes aspectos: incluir restricciones de precedencia, tareas aperiódicas, y tareas con restricciones de recursos.

- Extender PlataTR para la simulación de tareas de tiempo real sobre sistemas multiprocesadores. Para lograr este objetivo, se debe tomar encuenta que la planificación de tareas de tiempo real sobre multiprocesadores puede ser desarrollada siguiendo alguno de los siguientes esquemas: el esquema *particionado* o el esquema *global*. Cuando se utiliza un esquema particionado, el control de admisión no solamente decide cuáles tareas deben ser aceptadas, sino también decide en que procesador (problema de asignacion) debe ejecutarse cada tareas del conjunto. En un esquema particionado todas las instancias de una tarea en particular son ejecutas sobre un mismo procesador. En contraste, en un esquema global todas las instancias son almacenadas en una sola cola global y, en cualquier instante de tiempo, los procesadores ejecutan las instancias de las tareas que posean las prioridades más altas.

  No existen herramientas graficas que modelen conjuntos de tareas de tiempo real, sobre los esquemas particionado o global. Por esta razón, consideramos importante esta extensión como trabajo futuro.

Apéndice A

Manual de Usuario

En este apéndice, se describe el manual de usuario de la herramienta de simulación PlataTR. De forma, general podemos decir que las funcionalidades de PlataTR son:

- **Especificación de tareas**. PlataTR permite introducir los parámetros temporales de las tareas de 2 formas: 1) En la modalidad estática, por medio de ventanas de diálogo ó cargando un archivo de texto, ó 2) En la modalidad dinámica, por medio de un archivo generado desde un Micro-Kernel de tiempo real desarrollado en la Sección de Computación del Cinvestav-IPN.

- **Ploteo del comportamiento de las tareas**. Presenta la visualización gráfica del comportamiento de las tareas mediante gráficas de Gantt.

A.1 Requerimientos

PlataTR está desarrollado en el entorno de programación de Visual C++ 6.0 sobre la plataforma de Windows 2000. La herramienta es compatible con cualquier computadora personal, IBM o compatible, que utilize la tecnologia PENTIUM de Intel. En la ejecución de PlataTR, se recomienda utilizar al menos una computadora Pentium III a 550 MHz con 256 en RAM.

Para el desarrollo, se optó por el entorno de desarrollo de Visual C++ debido a que combina la programación orientada a objetos (POO) y el sistema de desarrollo para crear aplicaciones gráficas para windows (Software Development Kit, SDK).

A.2 Estructura del directorio

Los archivos del proyecto se encuentran almacenados en el directorio *\PlataTR*, el cual contiene los siguientes archivos.

Archivos Generales:

- *Plata.dsw y Plata.dsp*: Contiene los archivos que automatizan la construcción del proyecto. Plata.dsw (*developer studio workspace*) Se utiliza para que Visual C++ cargue el proyecto, y Plata.dsp (*developer studio project*) Es un archivo que contiene las órdenes necesarias para ejecutar los pasos requeridos para obtener el archivo ejecutable correspondiente de PlataTR.
- *Plata.h*: Este archivo contiene la cabecera principal de PlataTR. El archivo Plata.h incluye otros ficheros de cabecera necesarias para la aplicación, asi como la clase *CPlataApp*.
- *Plata.cpp*: Este archivo contiene la definición de la clase CPlata.
- *Plata.rc*: Este archivo contiene una lista de todos los recursos que PlataTR utiliza, como son los iconos, y los mapas de bits. Estos se guardan en el subdirectorio *\PlataTR\res*.
- *Plata.clw*: Este archivo contiene información utilizada por *ClassWizard*[1].

Archivos vinculados con las ventanas:

- *MainFrm.h* y *MainFrm.cpp*: Estos archivos contienen la declaración y definición de la clase *CMainFrame*, la cual se deriva de *CMDIFrameWnd*. Esta clase permite controlar el formato de la ventana raiz de PlataTR.

[1]ClassWizard es una herramienta del entorno de desarrollo de visual C++, que permite generar automáticamente las clases y funciones que gestionan los mensajes de los objetos Windows. Se utiliza para relacionar los controles de la aplicación con el código que se ha de ejecutar al actuar sobre dichos controles.

- *ChildFrm.h* y *ChildFrm.cpp*: Estos archivos declaran y definen *CChildFrame*, la cual se deriva de *CMDIChildFrame*. Esta clase permite controlar las caracteríticas de la ventana hija.

Archivos vinculados con el planificador:

- *ClassArr.h*: Este archivo contiene la clase genérica para el manejo de arreglos dinámicos.
- *Str.h*: Este archivo contiene la declaración de la clase CStr, cuyo objetivo es el manejo de las cadenas.
- *MisDatos.cpp* y *MisDatos.h*: Estos archivos definen la clase *CMisDatos*. En la clase CMisDatos se definen los parámetros temporales de las tareas y otros parámetros opcionales.
- *ColorBtn.cpp* y *ColorBtn.h*: PlataTR utiliza un esquema de colores con la finalidad de identificar las tareas en el momento de la visualización gráfica. Por lo que estos archivos definen la forma de seleccionar los colores.
- *RealTime.cpp* y *RealTime.h*: Estos archivos son el núcleo del planificador. Contiene las funciones relacionadas con la planificación de tareas de tiempo real.
- *Graphics.cpp* y *Graphics.h*: Este archivo contiene las primitivas de los elementos gráficos, que son utilizados para la simulación.

A.3 Iniciando PlataTR

PlataTR fue desarrollada usando la plataforma Windows. Su ejecución se inicia seleccionando el archivo *Plata.exe* desde el explorador de Windows como se muestra en la Figura A.1.

Una vez que se ha iniciado PlataTR, el usuario esta listo para introducir un nuevo conjunto de tareas, cargar un archivo con parámetros temporales el cual fue creado desde PlataTR y fue guardado con la extensión (*.ptr), o bien cargar un archivo de eventos generado desde el Micro-Kernel de tiempo real y cuya extensión es (*.txt).

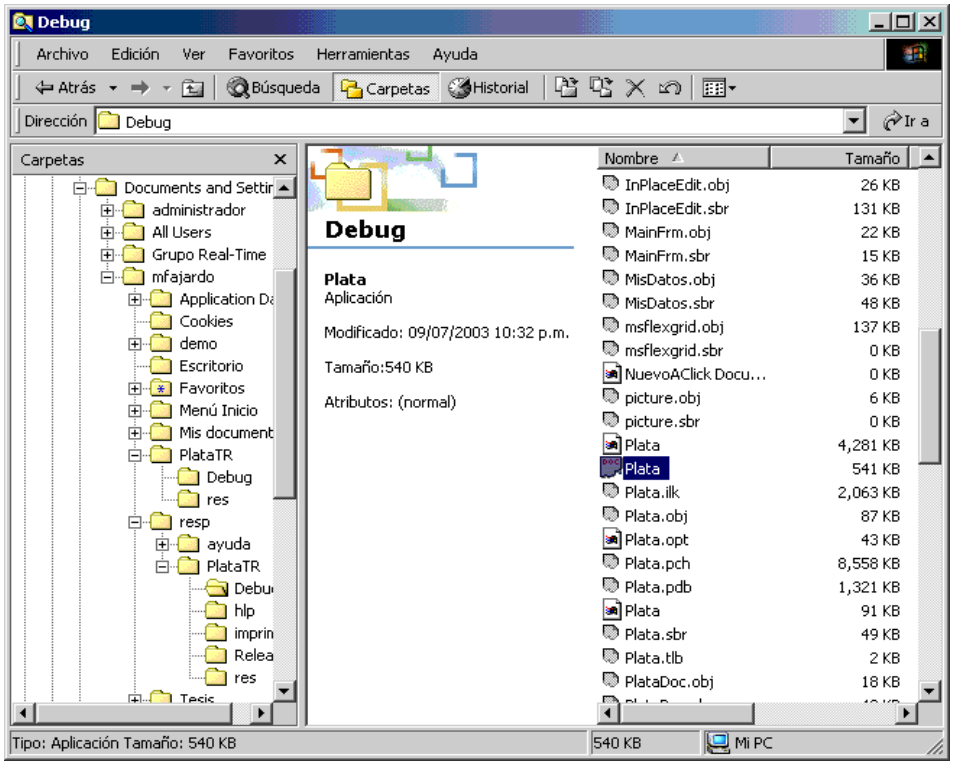

Figura A.1: Iniciando PlataTR desde el Explorador de Windows.

A.4 Manejo de PlataTR

A.4.1 Captura de un Conjunto de Tareas

Como se ha mencionando, PlataTR se puede utilizar de dos formas,

1. **Forma estática:**

   Consiste en crear un conjunto de tareas desde PlataTR. Para esto se selecciona el comando *Archivo->Nuevo* de la Barra de Menús de PlataTR. Una vez que se ha seleccionado dicha orden aparecerá una ventana denominada *Parámetros de entrada* como la que se muestra en la Figura A.2.

   La ventana de *Parámetros de entrada* de PlataTR asigna un número a la tarea que se ésta dando de alta. Así como también, calcula la utilización de procesador de dicha tarea. Los parámetros que puede introducir el usuario son:

   - **Descripción**. Este parámetro es opcional, sin embargo, es de gran ayuda para el

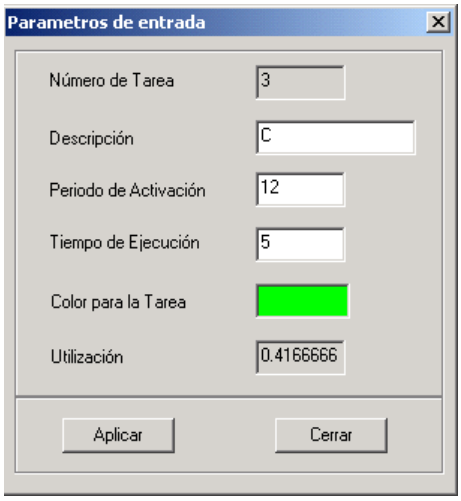

Figura A.2: Captura de parámetros temporales para crear un nuevo conjunto de tareas

usuario debido a que permite explicar textualmente la actividad que realiza cada tarea.

- **Período de activación**. Este parámetro es muy importante para llevar acabo el análisis y simulación de las tareas, el cual le permite a PlataTR identificar los instantes de activación de cada tarea.

- **Tiempo de cómputo**. De la misma manera que el parámetro anterior, este es obligatorio introducirlo, ya que con este parámetro se le indica a PlataTR el tiempo de cómputo que utiliza cada tarea para realizar su actividad en cada instancia.

- **Color para la tarea**. PlataTR a sido diseñada para facilitar la visualización del comportamiento de las tareas durante su ejecución, es por eso que a cada tarea se le puede asignar un color que la distinga de las demás. Al presionar el bóton *Color para la tarea* aparece la ventana de color de Windows la cual permite al usuario seleccionar un color o personalizar un color para la tarea que se esta dando de alta.

Para el caso de los parámetros obligatorios (*Período de activación* y *Tiempo de cómputo*), en caso de introducir el tiempo de cómputo mayor al período de activación PlataTR enviará al usuario un mensaje de error como el que se muestra en la Figura A.3.

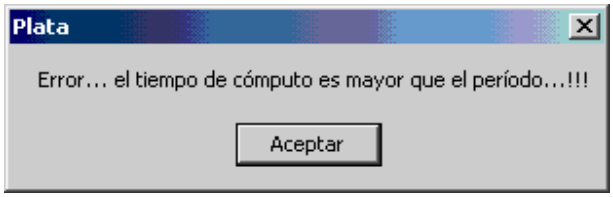

Figura A.3: Ventana de mensaje de error

2. **Forma Dinámica:**

   En la forma dinámica, las tareas se introducen a PlataTR mediante un archivo de eventos el cual fue generado por el Micro-Kernel de tiempo real. Para ello, el usuario debe seleccionar el comando *Archivo->Abrir->Dinámico* de la Barra de Menús de PlataTR.

   En este menú, el usuario deberá seguir las siguientes acciones:

   - Seleccionar el archivo con el conjunto de tareas que se desea analizar.
   - Seleccionar el comando *Algoritmos* de la Barra de Menús de PlataTR para seleccionar el botón con la etiqueta *Eventos del Micro-Kernel*.
   - Finalmente, el usuario debe presionar el botón *Aplicar* lo que dará inicio con la visualización gráfica de los eventos del Micro-Kernel de tiempo real.

A.4.2 Editar

PlataTR permite realizar las operaciones modificar los parámetros temporales, insertar o eliminar tareas. Para llevar acabo alguna de estas actividades el usuario debe seleccionar el comando *Editar* de la Barra de Menús de PlataTR. Esta acción invoca a la ventana de edición denominada *Conjunto de Tareas*, como se muestra en la Figura A.4.

La modificación de algún parámetro temporal se realiza haciendo doble click en la celda donde se ubica el parámetro a modificar. Debemos aclarar que existen parámetros los cuales no se pueden modificar como es el color, ubicación dentro del área de dibujo, o el número de tarea asignado al ser dada de alta la tarea.

Para insertar o eliminar una tarea solo se necesita hacer click en la columna con la etiqueta *Tarea* y en la fila donde se encuentra la tarea que se desea eliminar o el lugar donde se insertará la nueva tarea. La Figura A.4 muestra la selección de una fila, por el usuario. Como puede

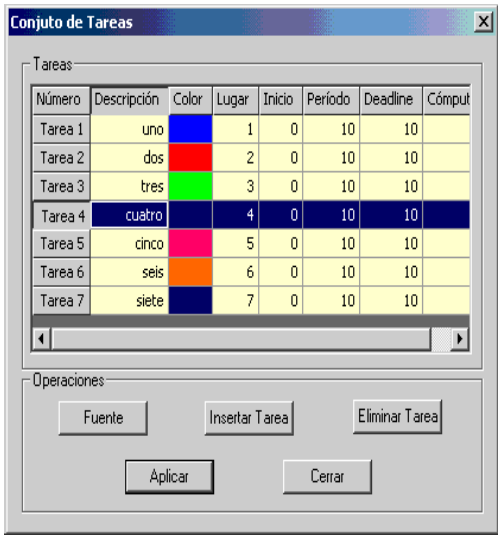

Figura A.4: Ventana de edición de tareas.

apreciarse la fila seleccionada obtiene un color diferente a las demás filas. Posteriormente, se presiona el botón *Insertar Tarea* o el botón *Eliminar Tarea*. Una vez que se insertó o eliminó una tarea es importante presionar el bóton *Aplicar*. De lo contrario no se realizará la modificación efectuada.

A.4.3 Ejecutando un algoritmo de planificación

Después que PlataTR tiene definido un conjunto de tareas, el usuario puede elegir un algoritmo de planificación mediante la vantana *Algoritmos de Planificación* como la que se muestra en la Figura A.5. Para obtener esta ventana, el usuario debe seleccionar el comando *Algoritmos* de la Barra de Menús de PlataTR. Posteriormente, el usuario debe presionar el botón *Aplicar* lo que permitirá dar inicio a la planificación de tareas.

A.4.4 Detener y Reanudar la Ejecución

PlataTR realiza la simulación gráfica utilizando animación. Esta animacion se realiza mediante el uso de ventanas deslizantes.

El usuario puede detener la ejecución para apreciar el comportamiento de las tareas. Para esto, el usuario debe seleccionar el comando *Ver->Detener* de la Barra de Menús de PlataTR.

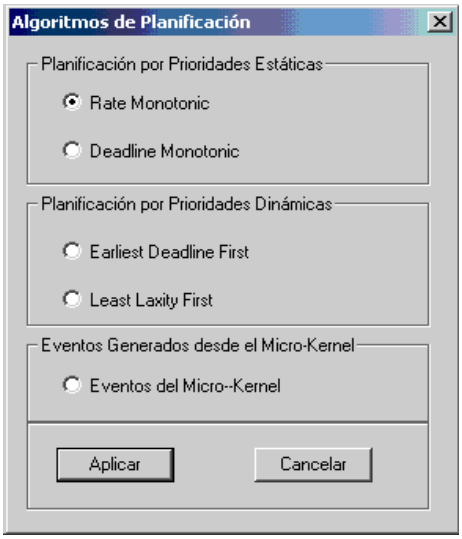

Figura A.5: Algoritmos de planificación de PlataTR

Para que la planificación continúe el usuario debe seleccionar el comando *Ver->Reanudar* desde la Barra de Menús de PlataTR.

Bibliografía

[1] IEEE Standard 896.1-1991. Standard for futurebus+logical protocol specification. *IEEE Computer Society*, March 1992.

[2] Stankovic John A. Misconceptions about real-time computing - a serious problem for next-generation systems. *IEEE Computer*, 21(10):10–19, October 1988.

[3] Paolo Ancilotti, Giorgio Buttazzo, Marco Di Natale, and Marco Bizzarri. A flexible tool kit for the development of real-time applications. In *Procceding of the 2th IEEE Real-Time Technology and Applications Symposium*, pages 260–262, Boston, Massachusetts, June 1996.

[4] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Ellings. Hard real-time scheduling: The deadline monotonic approach. In *Procceding 8th IEEE Workshop on Real-Time Operating Systems and Software*, May 1991.

[5] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Welling. The stress hard real-time system simulator. *Software-Practice and Experience*, 24(6):543–564, June 1994.

[6] Giorgio C. Buttazzo. *HARD REAL-TIME COMPUTING SYSTEMS Predictable Scheduling and Applications*. Kluwer Academic Publisher, 1997.

[7] Center for Computing Technology. *ASSERTS: User's Manual*, 1996.

[8] Francisco Javier Cevallos. *Microsoft Visual C++ 6 Aplicaciones para Win32*. Alfaomega Ra-Ma, second edition, 2000.

[9] TimeSys Corporation. http://www.timesys.com.

[10] David Decotigny and Isabelle Puaut. Artisst: An extensible framework for the simulation of real-time systems. Technical report, IRISA, Campus de Beaulieu, 35042 Rennes Cédex, France, 2001.

[11] Kanad Ghose, Sudhir Aggarwal, Pavel Vasek, S. Chadra, Amritansh Raghav, Abhrajit Ghosh, and David R. Vogel. Asserts: A toolkit for real-time software design, development and evaluation. *IEEE Computer*, pages 224–232, April 1997.

[12] Lui Sha John Lehoczky and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. *IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[13] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Procceding of the 10th IEEE Real-Time Symposium*, pages 166–171, December 1989.

[14] J. Y. T. Leung and J. Whiteheaad. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, December 1982.

[15] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[16] C. L. Liu and W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.

[17] Microsoft MSDN. http://www.msdn.microsoft.com/visualc.

[18] Baker T. P. and Alan Shaw. The cyclic executive model and ada. In *Proceding of the IEEE Real-Time Systems Symposium*, pages 120–129, December 1988.

[19] Jane W. S., Juan-Luis Redondo, Zhong Deng, Too-Seng Tia, Riccardo Bettati, Ami Silberman Matthew Storch, Rhan Ha, and Wei-Kuan Shih. Perts: An prototyping environment for real-time systems. Technical report, Department of computer Sciense University of Illinois Urbana, May 1993.

[20] Liu J. W. S., K. J. Lin, W. K. Shih, A. C. Yu J.and Y. Chung, and W.Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, pages 58–68, May 1991.

[21] Fabricio Sensini, Giorgio C. Buttazzo, and Paolo Ancilotti. Ghost: A tool for simulation and analysis of real-time scheduling algorithms. In *Procceding of the IEEE Real-Time Educational Workshop (RTEW'97)*, pages 42–49, Montreal Canada, June 1997.

[22] L. Sha and Gooddenough. Real-time scheduling theory and ada. *IEEE Computer*, pages 53–62, April 1990.

[23] David B. Stewart. A tool for analyzing and fine tuning the real-time properties of an embedded system. *IEEE Transactions on Software Engineering*, 29(4):311–326, April 2003.

[24] Oscar Miranda Gomez y Pedro Mejía Álvarez. Kernel de tiempo real para control de procesos. *Conferencia de Ingeniería Eléctrica*, Septiembre 2003.