MODELAMIENTO FORMAL DE SISTEMAS CONTINUOS DE CONTROL INDUSTRIAL EN AMBIENTES DISTRIBUIDOS

NORBERTO MUÑOZ RICO

CORPORACION UNIVERSITARIA AUTÓNOMA DE OCCIDENTE
DIVISIÓN DE INGENIERÍAS
UNIDAD ACADÉMICA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2003

MODELAMIENTO FORMAL DE SISTEMAS CONTINUOS DE CONTROL INDUSTRIAL EN AMBIENTES DISTRIBUIDOS

NORBERTO MUÑOZ RICO

Tesis de grado para optar al título de

Ingeniero Electrónico

Director
DIEGO MARTÍNEZ
Jefe del área de electrónica digital

CORPORACION UNIVERSITARIA AUTONÓMA DE OCCIDENTE
DIVISIÓN DE INGENIERÍAS
UNIDAD ACADÉMICA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
SANTIAGO DE CALI
2003

Nota de Aprobación

Trabajo aprobado por el comité de grado en cumplimiento de los requisitos exigidos por la Corporación Universitaria Autónoma de Occidente para optar al título de Ingeniero Electrónico.

Firma del presidente del jurado

EDUARDO VICTORIA
Firma del Jurado

JUANCARLOS MENA
Firma del jurado

Santiago de Cali, 24 de Julio 2003.

Esta tesis va dedicada a mi Madre quien con su esfuerzo me dio la educación, me apoyó mucho durante mi carrera y ha estado conmigo a todo momento.

AGRADECIMIENTOS

Por sobretodo le doy gracias a ***Dios*** que me iluminó, me dio paciencia y me facilitó todo para culminar exitosamente mi tesis de grado.

Tengo también muchas personas que me ayudaron en el desarrollo de mi proyecto de grado. Mencionaré solo algunos.

Agradecimientos especiales a *Gilberto Villa* (Ex-Auxiliar de Salas de Electrónica). El programa para realizar las validaciones, Design/CPN, trabaja bajo linux (entre los sistemas operativos más conocidos) y fue Gilberto la persona que me ayudo en esa parte haciéndome perder el miedo a este sistema operativo. Lastima que la CUAO no cuente en este momento con sus servicios.

Mi director de tesis, a pesar de sus *mal-genios*, me colaboró mucho en el desarrollo de la tesis, brindándome los espacios necesarios en la Corporación, y ayudándome en la conceptualización. Por esto, extiendo mi agradecimiento hacia él.

A mis compañeros de proyecto, Hugo Macias, Jorge Lizcano y Felipe Vivas, por brindarme la información requerida para realizar los modelamientos.

CONTENIDO

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE ANEXOS

Pág.

RESUMEN

La tesis está orientada a buscar la manera de reducir la complejidad que el diseño de un producto puede traer consigo, pero específicamente, se pretende aplicar las técnicas de diseño formal, en la especificación de sistemas continuos de control en ambientes distribuidos.

El diseño formal implica seguir una serie de pasos que dividan el proyecto en diferentes fases. En esta tesis se retoman algunas de las estrategias de diseño formal, para explicar en que nivel se encuentra el trabajo aquí propuesto. Después de tener claro lo que se desea realizar y tener unas especificaciones bien definidas, se llega a un nivel de diseño denominado *Nivel Arquitectural*, que es donde se utilizan ciertas estrategias para el modelamiento con el fin de optimizar el diseño, o evaluar la forma como este se puede plantear para posteriores etapas.

Para esta tesis el nivel arquitectural de diseño se aplica a sistemas continuos de control distribuido, para apoyar la aplicación del diseño formal a estos procesos. Es útil evaluar antes del montaje real, lo que se desea como resultado, y en ambientes distribuidos se necesita de técnicas especiales para tal fin, debido a que las plataformas sobre las que están implementadas todas las aplicaciones de los módulos, bien sea en software o en hardware, son en su mayoría diferentes. Dichas técnicas debe permitir realizar un análisis conjunto de todo el sistema, algunas de estas técnicas se mencionan en este texto.

INTRODUCCIÓN

Gran cantidad de procesos requieren la implementación de estrategias de diseño que permitan de una manera clara y concisa describir las necesidades u objetivos que se deben cubrir, para su posterior evaluación. Muchas veces los diseños se realizan sin tener en cuenta ningún parámetro o esquema de diseño, pasando directamente desde la necesidad hasta el diseño real, casos en los cuales las diferencias entre lo que se quería y lo que se desarrolló son notorias.

No todos los procesos exigen la utilización de formalismos para su ejecución, todo depende del tamaño del proyecto, la cantidad de requerimientos, plataforma de operabilidad, costos y otros factores que pueden afectar de manera directa o indirecta el diseño, y que son difíciles manejar y ser observados a todo momento del proceso.

Ejemplos de sistemas donde la necesidad de estrategias de diseño tienen validez y justificación explícita, son los procesos de control industrial. En dichos casos, se manejan variables que pueden presentar riesgos para los operarios o altos costos de implementación u operabilidad para la industria. Si en el momento de la etapa de diseño, no se manejan las variables de manera correcta, las consecuencias no tardarán en hacerse notar.

OBJETIVOS

OBJETIVO GENERAL

Plantear un conjunto de componentes que permitan realizar de manera formal el modelamiento y validación de sistemas continuos de control industrial en ambientes distribuidos.

OBJETIVOS ESPECIFICOS

- Analizar la forma en que se comportan los componentes de un sistema continuo de control industrial en ambientes distribuidos

- Seleccionar herramientas, lenguajes y plataformas adecuadas para la representación de este tipo de modelos.

- Elegir un método orientado a objetos que permita describir de una forma simple la arquitectura y el comportamiento del sistema.

- Plantear componentes utilizando modelos orientados a objetos que permitan representar el comportamiento del sistema a implementar por parte del diseñador.

- Realizar la representación del comportamiento de cada componente utilizando Redes de Petri.

- Realizar el modelado de un sistema de control de nivel que contiene un sensor, un actuador y una interfaz de usuario interconectados a través de una red CAN.

- Analizar las características temporales y funcionales del modelo del control de nivel utilizando herramientas para el desarrollo de validaciones dinámicas de modelos con redes de Petri.

1. ESTRATEGIA DE DISEÑO

Para el diseño formal de sistemas electrónicos, se debe precisar el uso de esquemas que permitan reducir la complejidad del proyecto a realizar. Para esto se cuenta con metodologías que permiten dividir al proceso de elaboración en etapas, con menor dificultad que enfrentar el problema de lleno.

Por medio de esquemas y estrategias de diseño, se puede lograr entender el funcionamiento de un sistema, sin necesidad de realizar un montaje real. Esto es importante para disminuir, en lo posible, el gasto innecesario generado por la falta de análisis en etapas previas al desarrollo físico. Además las estrategias formales de diseño, permiten implementar modelos con los que es posible predecir y controlar el comportamiento del sistema ante muchos eventos. Sobre los modelos se pueden estudiar casos extremos que si son dejados para un posterior análisis sobre el modelo final, podrían resultar en incrementos de costos y de riesgos.

1.1 FLUJOS DE DISEÑO

Para realizar de una manera formal el diseño de un proyecto, electrónico o de otra clase, lo primero que se debe hacer es adoptar un flujo de diseño que permita planificar la realización de cada una de las fases del proyecto, desde la concepción de la idea, hasta la obtención del prototipo final[1]. Las etapas intermedias, permiten la modificación de la concepción inicial de la idea, y la pueden moldear a los recursos con que se cuentan y a los impedimentos que se puedan presentar.

Las etapas comunes de un flujo de diseño se plantean continuación[2]:

1-Planteamiento.
2-Especificaciones.
3-Diseño arquitectural.
4-Diseño lógico.
5-Diseño físico.
6-Fabricación.
7.Validación.

[1] TERES, Luis; TORROJA, Yago; LOCOS, Serafín y VILLAR, Eugenio. Lenguaje estándar de Diseño Electrónico VHDL. España : McGraw-Hill, 1998. 239 p.
[2] MARTÍNEZ, Diego. Notas de Clase (Microcontroladores II). Cali : Corporación Universitaria Autonoma de Occidente, 2002.

Estos pasos pueden ser diferentes según el caso, pero normalmente llevan este esquema.

1.1.1 Etapa de requisitos En esta etapa se presenta formalmente la idea para que sea estudiada técnica y económicamente. Esta fase inicial es importante para no perder ni tiempo ni dinero, en un proyecto que quizá no es viable. En esta etapa, el portador de la idea, es la persona más importante.

1.1.2 Etapa de especificaciones Luego de que se determina que el proyecto es factible técnica y económicamente, se procede a describir las funciones y objetivos que el proyecto aspira a cubrir. En esta etapa se debe describir muy bien la funcionalidad del producto final, se puede realizar la división del trabajo, y otras descripciones técnicas.

1.1.3 Etapa de diseño arquitectural Durante esta etapa se utilizan herramientas de diseño para describir el sistema en función de las especificaciones. Por lo general se realiza uno o varios modelos que permiten observar las características del sistema. Sobre dicho modelo se realizan pruebas, cambios y toda clase de análisis del comportamiento del sistema. Es una de las etapas más importantes y quizá más largas en el diseño. También se pueden dejar establecidas cierta clase de pruebas que se aplicarán a posteriores etapas del diseño.

1.1.4 Etapa de diseño lógico o detallado El modelo entregado por la etapa anterior es modificado o corregido si es necesario, con el fin de adecuarlo para las etapas posteriores. En este paso, se realiza una optimización del modelo para que cumpla con algunos parámetros de las herramientas utilizadas en adelante. Puede ser algo como la traslación del código del modelo a otro código en el cual será implementado realmente. También puede ser la reducción del número de componentes incorporados en el modelo, aumento o disminución de retardos, etc.

1.1.5 Etapa de diseño físico Se realiza el análisis del sistema de manera real, conectando físicamente los componentes del diseño, o realizando los pasos necesarios para que el sistema entre en funcionamiento. En esta etapa se analizan factores reales del comportamiento del sistema y se corrigen para la etapa de fabricación.

1.1.6 Etapa de fabricación Se implementa el modelo como quedará finalmente. Este proceso puede consistir en la grabación de código en PLDs o FPGAs, por ejemplo. A los prototipos obtenidos se les realiza un test de fabricación.

1.1.7 Etapa de validación Luego de tener el prototipo, se realizan las verificaciones correspondientes para observar si se cumplieron las metas propuestas, y si no se presentan diferencias entre lo que se solicitó y lo que se obtuvo. Además, en esta fase del

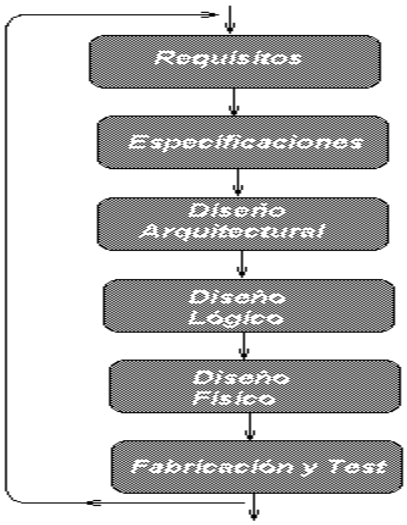
diseño se realiza la descripción final del producto para la documentación correspondiente, incluyendo las características, funcionales y físicas, originales y las que han sido adicionadas (si estas existen).

1.2 TIPOS DE DIAGRAMAS DE FLUJO

Luego de la división del proyecto en sus diferentes fases, se procede a seleccionar el flujo de diseño adecuado. La decisión del flujo de control útil debe realizarse teniendo en cuenta la relación existente entre cada una de las etapas del modelo. Así, los flujos de diseño se pueden presentar lineales estrictamente, lineales flexibles, paralelos, mixtos, etc.

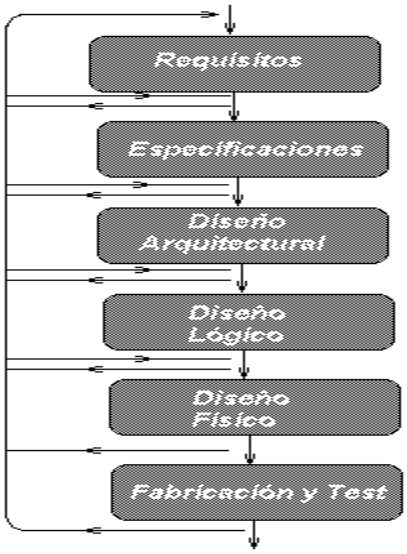
Los flujos lineales se caracterizan por la presentación de las etapas *una tras otra*, de tal manera que las tareas están relacionadas de manera secuencial. Algunos flujos lineales son implementados de tal manera que no permiten que se continúe con una etapa posterior hasta no completar la etapa que se encuentra en proceso (Figura 1); de esta forma se continua hasta la validación posterior a la fabricación, donde se decide si se cumplió o no con las expectativas.

Figura 1. Flujo de diseño lineal estricto



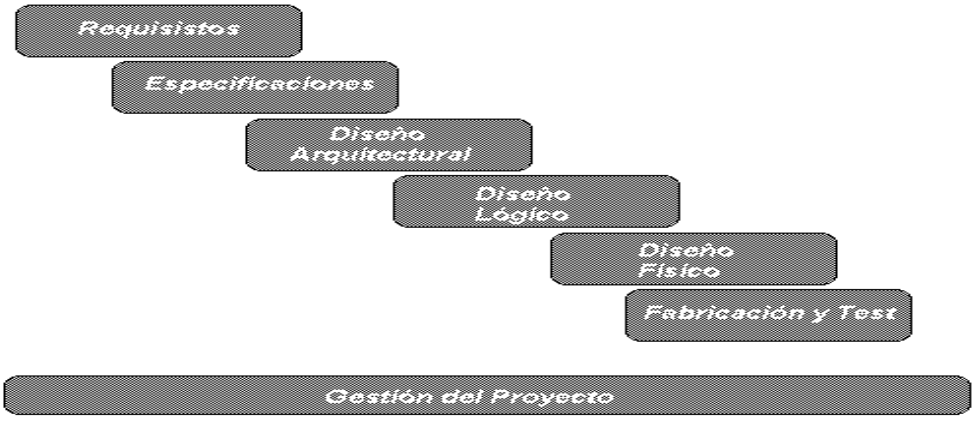
Otros flujos lineales permiten la evaluación de las diferentes etapas una vez son culminadas, y teniendo en cuenta las etapas previas se decide si esa etapa debe ser corregida, o si se debe retroceder a modificar otras etapas (Figura 2). Igualmente en otros modelos lineales se permite saltar pasos, y realizar otras maniobras de operación.

Figura 2. Flujo de diseño lineal flexible



Los flujos de diseño paralelos permiten la realización de diferentes pasos al tiempo, con lo cual se pueden estar analizando, corrigiendo o modificando las características del modelo inicial de forma simultánea. Estos modelos son particularmente útiles cuando el tiempo disponible es corto (Figura 3). La duración de cada etapa depende del proyecto.

Figura 3. Flujo de diseño paralelo



1.3 ARQUITECTO E INGENIERO

Cuando se encuentra definido el flujo de diseño, se puede realizar la asignación de tareas a cada integrante del grupo de trabajo.

Dejando de lado la generalidad de los flujos de diseño, y centrando la atención en el diseño de sistemas electrónicos, se puede observar que existen dos actores principales en la aplicación de estas estrategias de diseño: El Arquitecto y el Ingeniero.

El arquitecto es el encargado de reducir la complejidad, la incertidumbre y las ambigüedades a conceptos elaborables, mientras el ingeniero se concentra en hacer factibles esos[3].

El arquitecto se encarga de hacer un enlace entre el cliente o las necesidades y el proceso de diseño, realiza el primer acercamiento entre lo que se quiere y lo que se puede hacer. El ingeniero orienta toda la parte teórica entregada desde la parte arquitectural hacia el diseño físico, por medio de herramientas cuantitativas y cualitativas.

El trabajo arquitectural se centra en las etapas iniciales. El arquitecto toma los requerimientos del proyecto y los sintetiza por medio de herramientas de diseño, realizando con esto una especificación formal bien definida. El arquitecto, también puede realizar validaciones del modelo que realiza, para observar si se pueden satisfacer las necesidades del cliente, realizar reducciones en el diseño y en los costos, analizar y corregir las posibles causas de error del sistema y optimizar lo que mas pueda.

[3] Martínez, Op.cit.

2. DISEÑO ARQUITECTURAL

En los flujos de diseño, se menciona la etapa de diseño arquitectural, como la fase en la cual se establecen los parámetros que serán tenidos en cuenta para continuar con los siguientes pasos en el diseño. Es importante reconocer que se debe tener una clara descripción de lo que se realizará para que no se presenten problemas durante el diseño, por la incomprensión de lo que se solicitaba en principio. Después de definir las especificaciones del proyecto se necesita "traducir" estos requerimientos a un modo que permita interpretar sin ambigüedades y claramente el comportamiento del sistema a implementar.

2.1 CARACTERÍSTICAS DE UN DISEÑO ARQUITECTURAL

El diseño arquitectural surge de la necesidad de:

-Analizar las características de diversas clases de sistemas, incluyendo estructuras funcionales, de software y hardware principalmente.

-Encontrar los medios para detectar y manipular errores, además de conocer en que niveles del diseño deben ser implementados.

-Establecer políticas para la distribución y administración de recursos, realización de sincronizaciones, comunicaciones, planificación de tareas, etc.

Establecer un sistema arquitectural implica la consideración de factores críticos como costos, flexibilidad, seguridad y confiabilidad. Se presenta una dificultad en el hecho de que mientras algunas de estas propiedades son cuantificables, otras resultan imposibles de cuantificar, entonces la importancia de la arquitectura escogida se incrementa.

Un diseño arquitectural requiere del uso de modelos que describan las propiedades estructurales del sistema y su comportamiento. Estos modelos junto con las herramientas de análisis, permiten explorar los diferentes estados del sistema, y observar la disponibilidad en el tiempo de los recursos, como por ejemplo el hardware del sistema. También, con los modelos se puede recoger información que puede ser reutilizada en determinados casos.

A pesar de las bondades ofrecidas por el diseño arquitectural, cabe mencionar que surgen algunas desventajas, que pueden y deben ser tratadas con especial atención durante esta etapa del diseño:

-Se debe realizar un estudio sobre los componentes que serán utilizados dentro de los modelos a implementar.

-La evaluación es realizada de manera estática o a manera de modelo previo, o como una combinación de las características de los componentes, las redes y la arquitectura utilizada.

Esta tesis está enfocada a dar una estrategia para resolver estos dos inconvenientes en el modelamiento de sistemas continuos distribuidos en ambientes distribuidos.

Con el fin de soportar la fase de diseño arquitectural, y por tanto, resolver los problemas mencionados, se presentan algunas herramientas de diseño, que ofrecen características especiales para realizar las tareas encomendadas a esta etapa de diseño, las cuales se mencionarán en el presente capítulo.

2.2 MODELAMIENTO Y FASE ARQUITECTURAL DE DISEÑO

Dentro del nivel arquitectural del diseño, se deben realizar modelos. Con este fin son utilizadas a menudo representaciones graficas realizadas como entidades con atributos que permiten implementar relaciones funcionales propias del sistema. Es particularmente útil que dichas representaciones sean realizadas por medio de estructuras jerárquicas, ya que éstas permiten visualizar por niveles el contenido del sistema modelado, y las relaciones internas y externas de las distintas capas.

Algunas de las propiedades que deben ser cubiertas por medio de la representación arquitectónica de un modelo, son las siguientes:

-Propiedades Estructurales: Representación de los componentes del sistema, agrupación, e interacción entre ellos.

-Propiedades Extra-Funcionales: Demostración de que el sistema cumple con características o requisitos no especificados, pero necesarios del sistema: Capacidad, seguridad, flexibilidad, etc.

Para cumplir con las exigencias de un modelo arquitectural, en ocasiones no es suficiente con realizar un solo esquema de representación del sistema. Generalmente se plantea la necesidad de implementar distintos modelos con fines diferentes, pero representando en conjunto la funcionalidad del sistema en todos sus aspectos. Algunos de estos modelos pueden ser:

-Modelos Arquitecturales: Representación organizada de componentes.

-Modelos Dinámicos: Permite observar la evolución del sistema en función de determinados eventos.

-Modelos de Procesos: Se encargan de enfocar las aplicaciones especificas que se le darán al sistema modelado.

-Modelos Funcionales: Utilizados generalmente para realizar una representación jerárquica funcional del sistema.

Ejemplos de arquitecturas que implementan varias de las metodologías de diseño indicadas anteriormente son las arquitecturas centradas en datos, arquitecturas centradas en flujos de datos, arquitecturas de llamada y respuesta, arquitecturas orientadas a objetos, arquitecturas en capas, etc. Cada uno de estos ejemplos presenta características especiales para modelar distintos tipos de sistemas. Para el desarrollo de esta tesis de grado fueron escogidas las arquitecturas orientadas a objetos, porque cuentan con propiedades especiales para el modelamiento de sistemas distribuidos, y representan una buena opción al momento de analizar el problema en general, gracias a la incorporación de estructuras jerárquicas. En comparación con algunos de los otros ejemplos citados, los modelos orientados a objetos no centralizan la información (principio fundamental para la representación de sistemas distribuidos), permitiendo una relación más libre entre los diferentes componentes modelados en el sistema.

2.3 MODELOS ORIENTADOS A OBJETOS

Los modelos orientados a objetos, permiten realizar una descripción del sistema de manera muy completa, representando en niveles de abstracción las características del sistema. Los componentes del sistema encapsulan datos y operaciones, utilizadas para la comunicación entre objetos, la cual es realizada mediante el envío de mensajes.

Los atributos más importantes del diseño orientado a objetos son[4]:

Abstracción, ocultamiento de la información y encapsulamiento: Un objeto es definido por el servicio que presta a sus usuarios. El objeto cuenta con una interface que describe los servicios que este ofrece. Los detalles internos son descritos en procedimientos de Estructuras de Control de Operaciones y las propiedades en agentes de sincronización denominados Estructuras de Control de Objetos.

Descomposición Jerárquica: Existe una relación de desarrollo interno de objetos, es decir, existen objetos (denominados padres) que en su interior contienen otros objetos (conocidos como hijos). La relación jerárquica, o de parentesco, puede extenderse tanto como se necesite.

Estructuras de control: Las operaciones sobre los objetos son ejecutadas como flujos o hilos de control. Esto implica que varios hilos de control pueden operar simultáneamente sobre un objeto. La concurrencia es una de las principales características por las que los modelos orientados a objetos son ideales para el modelamiento de sistemas como los distribuidos.

Para el desarrollo modelamientos orientados a objetos, se han desarrollado diferentes lenguajes, o herramientas, que permiten la representación formal del diseño, guardando las características de esta estrategia de diseño. Uno de los principales estándares para el análisis orientado a objetos es UML, que ha reunido muchas características, que en la actualidad lo tienen como un lenguaje de modelamiento general. Otro tipo de estrategia de diseño es HRT-HOOD, que aunque no es tan general y tan práctico como UML, reúne propiedades que le dan importancia al momento de modelar sistemas en tiempo real.

Aunque con UML y HRT-HOOD se pueden analizar las características del sistema a nivel arquitectural, para lo que se desea realizar en esta tesis, se debe destacar que no son las dos únicas plataformas de modelamiento orientados a objetos; existen otras técnicas de modelamiento entre las que se encuentran[5]: Common Warehouse Metamodel (CWM™), Common Warehouse Metamodel (CWM™), Metadata Interchange Patterns (MIPS), Meta-Object Facility (MOF™), Software Process Engineering Metamodel (SPEM),
HOOD (No es critico para aplicaciones en tiempo real), ROOM, entre otros.

[4] BURNS, A y WELLINGS A.J. HRT-HOOD: A Structured Design Method for Hard Real-time Systems. Real-time and Distributed. Grupo de investigación en sistemas. Departamento de Ciencias de computación. Heslington, York : University of York, [sf]. 248 p.

[5] Object Management Group, inc (OMG) [En línea]. Marzo de 2003 [Citada en Abril de 2003]. Disponible en internet <http://www.omg.org/uml/>.

2.4 VALIDACIÓN EN EL DISEÑO ARQUITECTURAL

La evaluación de la disponibilidad y confiabilidad de los sistemas modelados formalmente constituyen dos de los factores más importantes en la etapa de diseño arquitectural.

Los factores a tener en cuenta dentro del análisis funcional de sistemas distribuidos y de control son: La disponibilidad, confiabilidad y operatividad de los componentes, las características de la red escogida y la topología de las arquitecturas. Por medio de factores como estos, se facilita la selección de la arquitectura ideal a implementar en el diseño, ya que se pueden examinar distintas alternativas sin necesidad de realizar grandes inversiones en su estudio.

Para cumplir con las expectativas en el análisis de los sistemas planteados, no es suficiente con tener una visión estática de los modelos realizados. Se deben practicar análisis dinámicos del comportamiento del modelo, y esto es difícil de realizar utilizando solamente modelos orientados a objetos. A fin de cumplir tal fin, es posible utilizar alternativas como:

-Simulación: Permite realizar modificaciones del modelo, prueba de diferentes casos y otras opciones.

-Evaluación matemática formal: Por medio de la utilización de estructuras de ecuaciones, estadísticas, análisis matriciales, etc., se logra una descripción completa del sistema.

La simulación es muy utilizada, en vista de las múltiples posibilidades de estudio que sobre ella se pueden hacer. La segunda opción es muy útil para realizar análisis de sistemas con bajos niveles de complejidad para un diseñador común.

Otra ventaja de utilizar la simulación, es que ésta es bien soportada matemáticamente, y entrega al usuario una interfaz en la cual no se deberá enfrentar la matemática compleja que el modelamiento de un sistema puede traer consigo.

Las redes de Petri, son una alternativa muy buena para cubrir los requerimientos del análisis formal del comportamiento del sistema. Su fuerte fundamento matemático hace que esta herramienta sea cada día más aceptada en el ambiente del modelamiento de sistemas.

Algunas de las razones por las que las redes de Petri son utilizadas por diferentes proyectos se presentan a continuación[6]:

-Posibilidad de construir modelos para diferentes clases de sistemas (en tiempo discreto o continuo, híbridos, complejos).

-Numero de extensiones existentes (coloreadas, estocásticas) permitiendo la representación de varias partes de los sistemas bajo diferentes puntos de vista.

-Existencia de herramientas que permiten desarrollar, simular y analizar modelos.

-Permite observar diferentes características como el control y la confiabilidad en un solo modelo, por medio de otras estrategias, se deben realizar análisis independientes.

-La posibilidad de observar la estructura conjunta del sistema. Por ejemplo el análisis de la red y la relación con otros componentes del modelo.

Teniendo en cuenta las propiedades de los componentes que se van a modelar en este documento, las ventajas especificas de las redes de Petri son:

-Modelamiento de ambientes distribuidos: Permiten modelar concurrencia, secuencias, conflictos y comunicación entre hilos de control.

-Independencia de tareas: Al momento de modelar el software del sistema se encuentran tareas independientes y dependientes entre sí. Una de las condiciones más importantes para los algoritmos de planificabilidad más conocidos, como DeadLine y Rate Monotonic, es la independencia de las tareas. Esto implica que estos algoritmos no permiten el análisis de sistemas que utilizan componentes para comunicación en forma excluyente, tal es el caso de objetos protegidos.

-Modelamiento de tareas protegidas: La planificabilidad del sistema por estrategias tradicionales de modelamiento (algoritmos matemáticos) se ve afectada por la utilización de tareas protegidas. Cuando una tarea protegida es asignada al procesador, éste no será liberado hasta el final de la ejecución de dicha tarea. Como consecuencia de esta condición la planificación realizada puede verse afectada.

[6] JENSEN, Kurt. An Introduction to the Theoretical Aspects of Coloured Petri Nets. Dinamarca, 1991. 56 p.

En base a estas características, como método de validación de los componentes que se describirán en el capítulo 5, serán utilizadas las redes de Petri.

3. DEFINICIONES: UML, HRT-HOOD Y REDES DE PETRI

La importancia del modelamiento de sistemas por medio de lenguajes o herramientas de diseño, radica en la pluralidad de entornos de desarrollo sobre los cuales se puede implementar una aplicación. Es decir, se requiere de normalizaciones, en lo posible con fundamentos matemáticos, que permitan describir las características de los diferentes componentes del sistema a modelar y la relación entre ellos.

En el ambiente del modelamiento aparecen diferentes métodos para representar sistemas, tales como los diagramas de estados, diagramas de flujo, y otros, que a su modo y con sus propiedades facilitan esta tarea. Dependiendo del tipo de sistema que se pretende modelar, se debe escoger la estrategia a utilizar para realizar el modelo.

Para realizar la descripción de componentes a nivel arquitectural, se presentaron anteriormente los modelos orientados a objetos como la opción indicada para la representación de sistemas distribuidos. Se mencionaron a UML y a HRT-HOOD, como dos herramientas de diseño apropiadas para la implementación de los diferentes componentes que se pretenden modelar. Además se planteó la utilización de las redes de Petri, como herramienta de validación formal del diseño. Este capítulo contiene una introducción breve al marco conceptual que dichas teorías contienen, pero especialmente enfocado a su utilización.

3.1 LENGUAJE DE MODELAMIENTO UNIFICADO: UML

El Lenguaje de Modelamiento Unificado es utilizado para especificar, visualizar, construir, y documentar cada una de las partes que comprende el desarrollo de sistemas. UML entrega una estrategia para modelar elementos conceptuales como lo son los procesos de negocio y funciones de sistemas, además de aplicaciones concretas como por ejemplo, escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reutilizables.

En estos momentos UML se ha convertido en un estándar de diseño orientado a objetos, en especial porque ha reunido, o es producto, de tres de las mejores técnicas para el análisis orientado a objetos: Booch OOD (Object-Oriented Design), Rumbaugh OMT (Object Modeling Technique) y Jacobson OOSE (Object-Oriented Software Engineering).

La utilización de UML es independiente del lenguaje de programación y de las características de los sistemas, ha sido diseñado para simplificar el desarrollo de sistemas de cualquier rama.

Como cualquier lenguaje, UML tiene su propia notación y sintaxis. Algunas de las principales ventajas de un lenguaje con una semántica bien definida, son las siguientes:

-Claridad: Puede actuar como punto de referencia para resolver ambigüedades en algunas interpretaciones y para despejar cualquier confusión en el significado preciso de un modelo.

-Equivalencia y Consistencia: La semántica provee bases claras, con las cuales se puede comparar y contrastar a estos lenguajes con otras técnicas o notaciones, y es posible también asegurar la consistencia entre sus diferentes componentes.

-Extensibilidad: La solidez de las extensiones del lenguaje pueden ser verificadas (a menudo es promovida por los autores).

-Refinamiento: Los pasos del diseño implementado en UML pueden ser verificados y documentados correctamente. Un desarrollo semánticamente correcto soporta modificaciones en su diseño, en las cuales un modelo puede ser dramáticamente transformado durante su implementación.

-Pruebas: Las pruebas justificadas y los análisis rigurosos para un sistema descrito en UML requieren una semántica precisa. Muchos de estos análisis no pueden ser realizados en UML.

-Posible implementación de Herramientas: Para el desarrollo de herramientas como generadores de código o verificadores de sintaxis , es imprescindible una semántica muy clara.

Muchos de estos beneficios no pueden ser alcanzados por UML, ya que su semántica no es lo suficientemente formal. Pero esto no ha sido impedimento para permitir que UML sea ampliamente utilizado, además de que los resultados obtenidos en pruebas realizadas han entregado resultados satisfactorios, y ponen de manifiesto que UML es una muy buena herramienta en el soporte del diseño arquitectural.

3.1.1 Clasificación de los diagramas de UML Dentro del ambiente de UML, se divide el proyecto en varios diagramas que representan las diferentes vistas del proyecto. Estos diagramas juntos constituyen la arquitectura del sistema. Con UML se logra una representación estática y dinámica del sistema, con lo cual se pueden observar, en la fase de diseño, errores o necesidad de sincronizaciones, por ejemplo.

En UML se presentan doce tipos de diagramas, clasificados en tres categorías [4]: Los diagramas que representan la estructura estática son cuatro, para la representación dinámica se utilizan cinco diagramas y tres diagramas representan las diferentes maneras en que se pueden manejar y organizar los módulos de la aplicación.

Diagramas Estructurales: Diagrama de clases, diagrama de objetos, diagrama de componentes y diagrama de despliegue.

Diagramas comportamentales: Diagrama de casos de uso, diagrama de secuencia, diagrama de actividades, diagrama de colaboración y diagrama de estados.

Diagramas de manejo del modelo: Paquetes, subsistemas y modelos.

A continuación se dará una breve descripción de los diagramas estáticos y dinámicos (más adelante se profundizará un poco en los diagramas más utilizados)[7].

-Diagramas estáticos

Diagrama de Clases: Modelan estructuras de clases y contenidos usando elementos de diseño tales como clases, paquetes y objetos. Estos diagramas muestran relaciones tales como contenidos, herencias, asociaciones y otras. Entregan una vista estática del proyecto.

Diagrama de Objetos: Se presentan las instancias de las clases modeladas en el diagrama de clases, y las relaciones entre ellas.

Diagrama de Componentes: Muestra a un alto nivel la estructura empaquetada del código. La cantidad de dependencias de los componentes también son modeladas, incluyendo códigos fuentes, componentes de código binario y componentes ejecutables. En general los diagramas de componentes plantean la organización de los componentes del sistema. Un componente se relaciona con una o varias clases, interfaces o colaboraciones.

Diagrama de Despliegue: Permite observar la relación existente entre los conjuntos de elementos o nodos, con el fin de disminuir la complejidad, presentada en algunas ocasiones, de los diagramas de clases y componentes.

[7] Embarcadero Technologies, inc [En línea]. [San Francisco, U.S.A], 2003 [Citada en febrero 2003]. Disponible en internet <http://www.embarcadero.com/support/what_is_uml.asp>.

-**Diagramas dinámicos**

Diagrama de Casos de Uso: Muestran la relación entre los actores y las acciones (casos de uso). Determinan lo que se puede realizar en el modelo.

Diagrama de Secuencias: Presenta la secuencia de tiempo de los objetos participantes en la interacción. Consiste de la dimensión vertical (tiempo) y de la dimensión horizontal (diferentes objetos).

Diagrama de Actividades: Modela un diagrama de estado especial en el que la mayoría de estados y transiciones son provocados por la finalización de las acciones en los estados iniciales. Este diagrama se enfoca en flujos controlados por procesos internos.

Diagrama de Colaboración: Muestra una interacción organizada de los objetos y sus enlaces. Son utilizados números para mostrar la secuencia de los mensajes. Junto con los diagramas de Secuencias, se conocen como *diagramas de interacción*.

Diagrama de Estados: Permiten observar la secuencia de estados, junto con sus respuestas y acciones, que un objeto puede tener durante su vida en reacción a estímulos recibidos. Son especialmente útiles para sistemas que reaccionen a eventos.

Por lo general no todos los diagramas son utilizados en los proyectos, todo depende del sistema que se va a modelar. Para diferentes aplicaciones resulta útil la siguiente propuesta:

Aplicación monopuesto:
-Diagrama de casos de uso
-Diagrama de clases
-Diagrama de secuencia
-Diagrama de colaboración

Aplicaciones monopuesto con entrada de eventos:
-Adicionar diagrama de estados

Aplicaciones cliente servidor
-Añadir diagrama de despliegue y diagrama de componentes, dependiendo de la complejidad

Aplicación compleja distribuida
-Todos los diagramas

3.1.2 Diagrama de clases Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenido. Estos diagramas solo dan una visión estática del sistema.

Un diagrama de clases esta compuesto por los siguientes elementos:

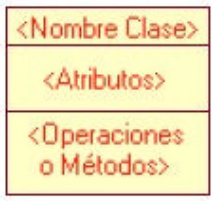
Clases: Atributos, métodos y visibilidad.
Relaciones: Herencia, composición, agregación, asociación y uso.

- Clases Es la unidad básica que encapsula toda la información de un objeto (un objeto es una instancia de una clase). A través de ella es posible modelar el entorno en estudio (una casa, un auto, una cuenta de ahorros, etc.).

Las clases en UML se representan como se muestra en la figura 4, donde se pueden observar los siguientes componentes:

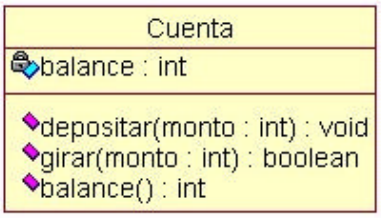
Nombre: Cada clase debe contener un único identificador o nombre.
Atributos: Los atributos representan las propiedades de la clase que pueden ser encontradas en todas las instancias de la clase. Estos atributos o variables de instancia caracterizan a la clase; pueden ser private, protected o public.
Métodos: Son las operaciones, las cuales interpretan la forma como interactúa el objeto con su entorno.

Figura 4. Formato de clase UML



Por ejemplo, una cuenta corriente tiene como característica: Balance, y puede realizar las siguientes operaciones: Depositar, girar y balance. El diseño para este caso es presentado en la figura 5.

Figura 5. Ejemplo de Clase en UML



Atributos

Los atributos de una clase se pueden clasificar en tres tipos, por medio de los que se manifiesta el grado de comunicación y visibilidad con el entorno:

public (+, ): Indica que el atributo será visible tanto dentro como fuera de la clase.

private (-, ): Señala que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden accesar).

protected (#, ): Advierte que el atributo no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de las subclases que se deriven.

Métodos

Los métodos de una clase son la manera como ésta interactúa con su entorno, sus características son:

public (+, ): Indica que el método será visible tanto dentro como fuera de la clase.

private (-, ): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden accesar).

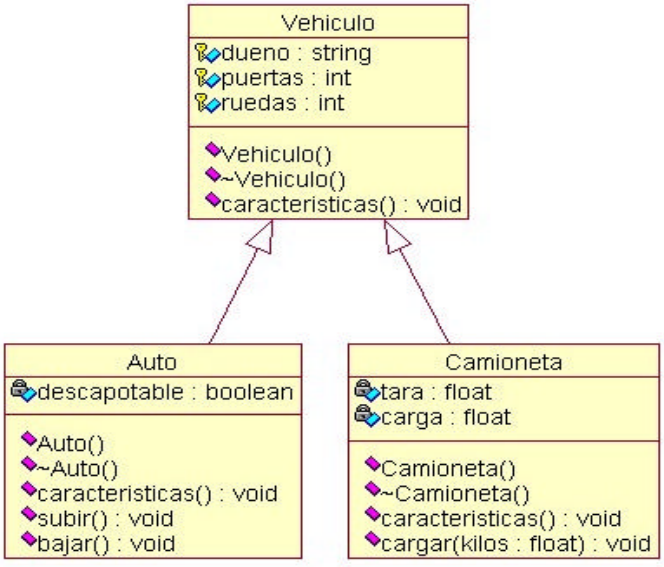
protected (#, ): Indica que el método no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de métodos de las subclases que se deriven.

- Relaciones. Indican la manera como se pueden interrelacionar las diferentes clases del modelo. Las relaciones enlazan una clase origen con una clase destino. A partir de la clase origen se realiza la acción a relacionar. La representación del sentido de la relación se realiza por medio de flechas, en las cuales, el lugar de donde parte la flecha es su clase origen y el sitio a donde llega es la clase destino. Los cuatro tipos de relaciones entre clase se describen a continuación:

a. Herencia (Especialización / Generalización):  Indica que una subclase hereda los métodos y atributos especificados por una *súper clase*, por consiguiente la subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la clase superior (public y protected).

Ejemplo.
En la figura 6, se especifica que Auto y Camión heredan de Vehículo, es decir, Auto posee las Características de Vehículo (precio, velocidad máxima, puertas, ruedas, etc) además posee algo particular que es *descapotable*, en cambio Camión también hereda las características de Vehiculo (precio, velocidad máxima, puertas, ruedas, etc) pero posee como particularidad propia el *acoplado*, *la tara* y *carga*.

Figura 6. Ejemplo de herencia de clases



Cabe destacar que fuera de este entorno, lo único "visible" es el método Características aplicable a instancias de Vehículo, Auto y Camión, pues tiene definición publica, en cambio atributos como Descapotable no son visibles por ser privados.

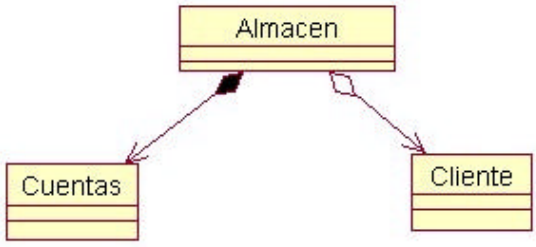
b. Agregación:  Para modelar objetos complejos, no bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y cadenas de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas anteriormente durante el desarrollo del proyecto, se tienen dos posibilidades:

-Por valor: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada *composición* (el objeto base se construye a partir del objeto incluido).

-Por Referencia: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es denominada a*gregación* (el objeto base utiliza al incluido para su funcionamiento).

Ejemplo.

Figura 7. Ejemplo de Agregación



En la figura 7, se destaca que:

-Un Almacén posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).

-Cuando se destruye el Objeto Almacén también son destruidos, los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.

-La composición (por Valor) se destaca por un rombo relleno.

-La agregación (por Referencia) se destaca por un rombo vacío.

La flecha en este tipo de relación indica la navegabilidad del objeto referenciado. Cuando no existe este tipo de particularidad la flecha se elimina.

c. Asociación: La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre si. Ésta no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

Ejemplo.

Figura 8. Ejemplo de Asociación.

Un cliente puede tener asociadas muchas Ordenes de Compra, en cambio una orden de compra solo puede tener asociado un cliente.

La cardinalidad expresa el grado y nivel de dependencia en una relación. Se citan en los extremos de la relación, de la siguiente forma:
-Uno o más: 1..* (1..n)
-Cero o más: 0..* (0..n)
-Número fijo: m (m denota el numero)

d. Dependencia o Instanciación (uso): Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase). El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo una aplicación grafica que instancia una ventana (la creación del objeto Ventana esta condicionado a la instanciación proveniente desde el objeto Aplicación), figura 9.

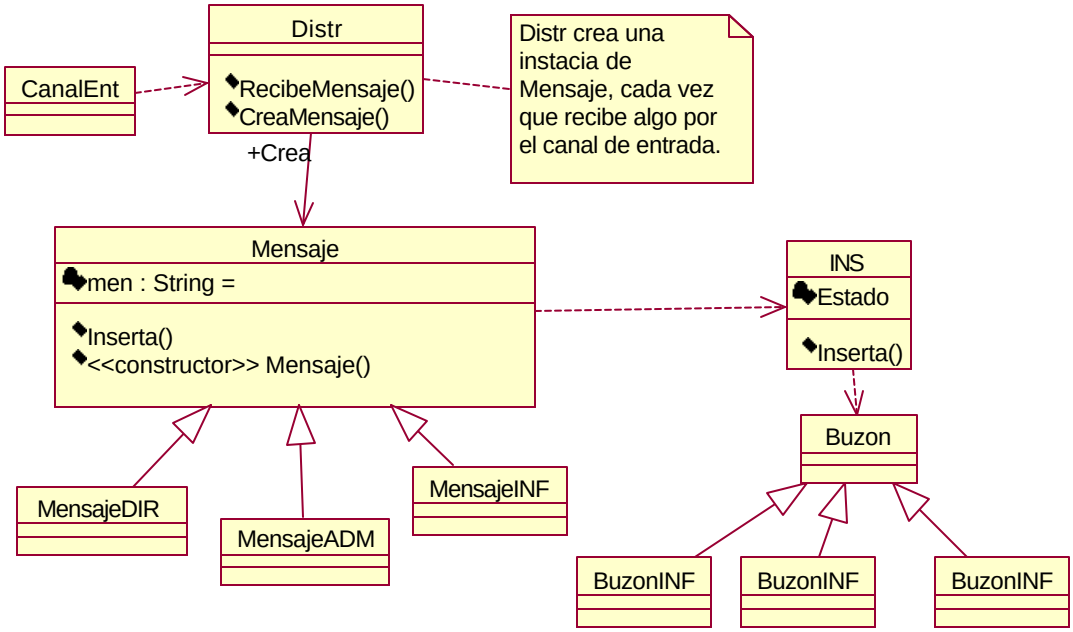
Figura 9. Ejemplo de Dependencia

Se debe tener en cuenta que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

3.1.3 Diagrama de objetos Por medio de este diagrama se pueden modelar las instancias de las clases del Diagrama de Clases. Representa a los objetos y sus relaciones en un momento dado.

En este diagrama se muestra un estado del diagrama de eventos. Para realizar el diagrama de objetos debe estar decidido que situación del sistema se desea representar. Por ejemplo, disponiendo de un sistema de mensajería, se toma la decisión de que el sistema se representará con dos mensajes entrantes, los dos para diferentes departamentos, dejando un departamento inactivo. Para el Diagrama de Clases de la figura 10, se tendría un diagrama de objetos con dos instancias de Mensaje, más concretamente con una instancia de MensajeDIR y otra de MensajeADM, con todos sus atributos valorados. También existiría una instancia de cada una de las otras clases que deban tener instancias. Como CanalEnt, INS, Distr, y el Buzón correspondiente a la instancia de mensaje que se esté instanciando. En la instancia de la clase INS se deberá mostrar en su miembro Estado, que esta ocupado realizando una inserción.

Figura 10. Diagrama de Clases para sistema de mensajería (Ejemplo).



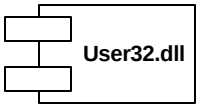
En un diseño pueden aparecer muchos diagramas de objetos, cada uno de ellos representando diferentes estados del sistema.

3.1.4 Diagrama de componentes Muestra la organización y las dependencias entre u conjunto de componentes. Por lo general el modelamiento de los componentes de un sistema se realiza por secciones.

En él se sitúan librerías, tablas archivos, ejecutables y documentos que formen parte del sistema.

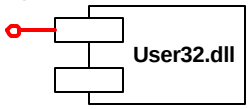
Puede servir para ver que componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema.

Figura 11. Componente del Sistema operativo Windows



En la figura 11, se observa un componente del sistema de Windows. En el diagrama de componentes de Windows debe salir este componente, ya que sin el sistema no funcionaría.

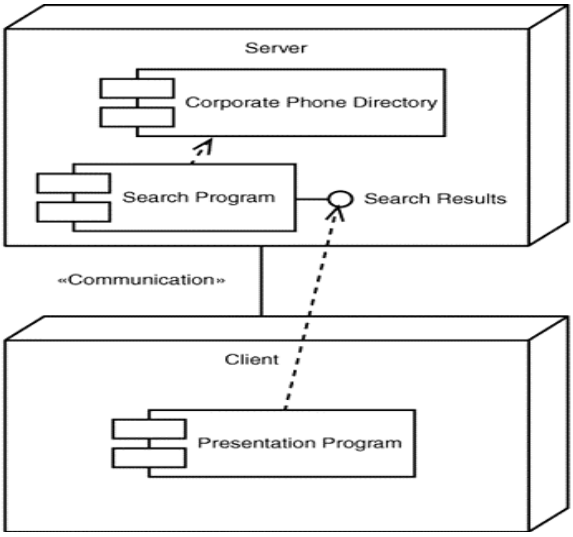
Figura 12. Componente del Sistema operativo Windows



En la figura 12, se muestra el mismo componente, pero indicamos que dispone de una interface. Al ser una *dll* la interfaz permite acceder a su contenido. Esto hace pensar que la representación anterior es incorrecta, pero no es así solo corresponde a un nivel diferente de detalle.

3.1.5 Diagrama de despliegue En el diagrama de despliegue se indica la situación física de los componentes lógicos desarrollados. Es decir se sitúa el software en el hardware que lo contiene. Cada Hardware se representa como un nodo.

Figura 13. Ejemplo de Diagrama de despliegue



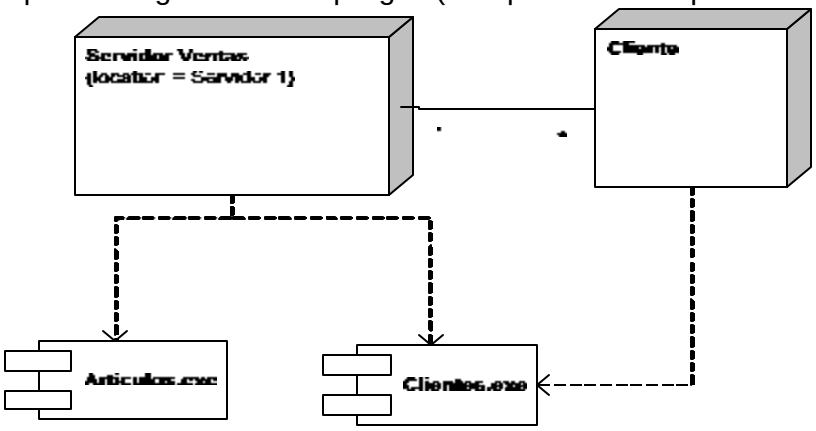
Un nodo se representa como un cubo, un nodo es un elemento donde se ejecutan los componentes, representan el despliegue físico de estos componentes.

En la figura 13, se tienen dos nodos, el cliente y el servidor, cada uno de ellos contiene componentes. El componente del cliente utiliza una interface de uno de los componentes del servidor. Se muestra la relación existente entre los dos Nodos. Se pueden asociar otras características (estereotipos) para indicar el tipo de conexión entre el cliente y el

servidor, así como modificar su cardinalidad, para indicar que soportamos diversos clientes.

Como los componentes pueden residir en mas de un nodo, se puede situar el componente de forma independiente, sin que pertenezca a ningún nodo, y relacionarlo con los nodos en los que se sitúa, figura 14.

Figura 14. Ejemplo de Diagrama de despliegue (Componentes independientes)



3.1.6 Diagrama de casos de uso Son útiles para visualizar el comportamiento del sistema, una parte de él o de una sola clase. Es muy útil para modelar partes de los sistemas, ya que solo se preocupa por el comportamiento y no por la forma de la implementación.

El diagrama de casos de uso representa la forma en como un cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso).

Un diagrama de casos de uso consta de los siguientes elementos:

-Actor.
-Casos de Uso.
-Relaciones de Uso, Herencia y Comunicación.

- Actor  Una definición previa, es que un Actor es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con

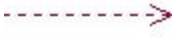
esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema.

Como ejemplo a la definición anterior, se puede considerar el caso de un sistema de ventas en que el rol del Vendedor con respecto al sistema puede ser realizado por un Vendedor o bien por el Jefe de Local.

- Caso de Uso  Es una operación o tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un Actor o bien desde la invocación desde otro caso de uso.

- Relaciones

a. Asociación:  Es el tipo de relación más básica que indica la invocación desde un Actor o Caso de Uso a otra operación (caso de uso). Dicha relación se denota con una flecha simple.

b. Dependencia o Instanciación:  Es una forma muy particular de relación entre clases, en la cual una clase depende de otra, es decir, se instancia. Dicha relación se denota con una flecha punteada.

c. Generalización:  Este tipo de relación es uno de los más utilizados, cumple una doble función dependiendo de su ambiente de aplicación, que puede ser de *Uso* (<<uses>>) o de *Herencia* (<<extends>>). Este tipo de relación esta orientado exclusivamente para Casos de Uso (y no para actores).

-Uses: Se recomienda utilizar cuando se tiene un conjunto de características que son similares en más de un Caso de Uso y no se desea mantener copiada la descripción de la característica.

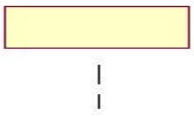
-Extends: Se recomienda utilizar cuando un Caso de Uso es similar a otro.

En el Apéndice A, se presenta un ejemplo de Diagramas de Uso de Casos.

3.1.7 Diagramas de interacción Los diagramas de interacción, representan la forma en como un Cliente (Actor) u Objetos (Clases) se comunican entre si en petición a un evento. Esto implica recorrer toda la secuencia de llamadas, de donde se obtienen las responsabilidades claramente. Dichos diagramas pueden ser obtenidos de dos partes, desde el Diagrama Estático de Clases o el de Casos de Uso.

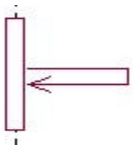
Los componentes de un diagrama de interacción son:

-Un objeto o Actor.
-Mensaje de un objeto a otro objeto.
-Mensaje de un objeto a sí mismo.



- Actor El rectángulo representa una instancia de un objeto en particular, y la línea punteada representa las llamadas a métodos del objeto.

- Mensaje a Otro Objetos Se representa por una flecha entre un objeto y otro, representa la llamada de un método (operación) de un objeto en particular.

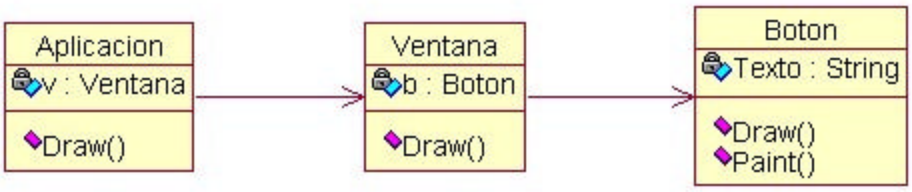


- Mensaje al Mismo Objeto No solo llamadas a métodos de objetos externos pueden realizarse, también es posible visualizar llamadas a métodos desde el mismo objeto en estudio.

Ejemplo.

A partir del modelo estático de la figura 15, se obtendrá el diagrama de interacción correspondiente.

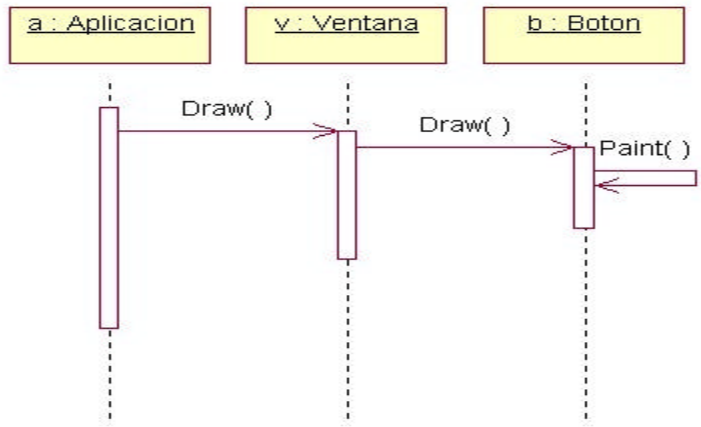
Figura 15. Diagrama de Casos para ejemplo de Diagrama de Interacción



Aquí se representa una aplicación que posee una Ventana gráfica, y ésta a su vez posee internamente un botón.

El diagrama de interacción para dicho modelo se presenta en la figura 16.

Figura 16. Ejemplo de Diagrama de Interacción.



En donde se hacen notar las sucesivas llamadas a Draw() (entre objetos) y la llamada a Paint() por el objeto Botón.

3.2 DISEÑO JERÁRQUICO ORIENTADO A OBJETOS PARA SISTEMAS CRÍTICOS EN TIEMPO REAL: HRT-HOOD

Esta metodología de diseño fue diseñada en base a HOOD (Diseño Jerárquico Orientado a Objetos), el cual plantea la organización de los objetos, basados en criterios como la Jerarquía, el flujo de control y las actividades.

UML plantea la utilización de objetos y sus relaciones (por medio de diagramas), para modelar gran cantidad de sistemas. El Lenguaje de Modelamiento Indefinido (UML) es una plataforma para el desarrollo de modelos muy general, lo que crea la necesidad de especificar algunos de sus aspectos para utilizarlo en aplicaciones que así lo requieran. Es así como HOOD aparece para realizar modelamientos que requieran darle principal importancia al diseño funcional del sistema.

HOOD cumple con sus objetivos reduciendo el campo tan amplio de objetos que se pueden modelar en UML, a solamente dos objetos: Pasivo y activo. Para soportar aplicaciones de tiempo real, estos dos objetos no eran suficientes, razón por la cual apareció otra extensión de UML, HRT-HOOD. Con esta nueva técnica se incrementó la cantidad inicial de objetos planteados en HOOD para cubrir aspectos no solo funcionales,

sino además temporales. HRT-HOOD propone la utilización de los objetos Pasivos y Activos, y aporta otros como los objetos Protegidos, Cíclicos y Esporádicos. Con estos objetos aparecen características de tiempo real que no poseía HOOD, tales como, prioridades, períodos, plazos, tiempos de ejecución, etc.

HRT-HOOD modela sistemas de tal manera que la representación, se puede simplificar por medio de objetos que a niveles jerárquicos interpreten toda la estructura funcional del sistema. Cuando se habla de jerarquía, a lo que se refiere es a que los objetos en una etapa de representación general, no permiten observar cada una de las características de su funcionamiento interno, solamente entregan al esquema de nivel superior una descripción de las actividades principales, y la forma en que interactúan entre objetos del mismo nivel. Luego, cuando un análisis más profundo es requerido, se puede ir interiorizando sobre cada objeto hasta tanto el modelo necesite una descripción.

3.2.1 Descripción de los objetos de HRT-HOOD Cada uno de los objetos tiene características propias que definen su comportamiento frente a ciertos eventos, o actividades internas o externas. A continuación se dará una breve descripción de las principales propiedades de los objetos de HRT-HOOD, tratando de establecer un marco comparativo entre algunas características (2).

- Objetos Pasivos. Estos objetos no tienen control sobre la ejecución de sus operaciones, cuando son invocadas por otros objetos. No puede invocar funciones de otros objetos deforma espontánea (no puede contener tareas). Cuando un flujo de control invoca una operación de la interfaz, el flujo actua directamente sobre la operación. Sirven para modelar recursos pasivos sin actividad propia.

- Objetos Activos Es el objeto más general en HRT-HOOD; no tiene restricciones. Controla la ejecución de sus operaciones cuando son invocadas. Puede invocar operaciones de otros objetos de manera espontánea. Cada objeto protegido puede tener su flujo de control; además pueden cederlo a objetos pasivos recobrándolo al final de su ejecución.

- Objetos Protegido Al igual que los objetos activos pueden controlar la ejecución de sus operaciones al ser invocados por otros objetos. No tiene pueden invocar operaciones de otros objetos de manera espontánea. Pueden establecer bloqueo a sus invocantes.

- Objetos Cíclicos Pueden hacer llamados a operaciones de otros objetos de manera espontánea. Modelan actividades periódicas.

- Objetos Esporádicos Espontáneamente pueden invocar operaciones en otros objetos. Representa actividades esporádicas. Tiene una operación que es ejecutada cuando el objeto es invocado.

- Otras características de HRT-HOOD Un sistema modelado en HRT-HOOD en su nivel terminal no pueden contener objetos Activos, ya que a estos objetos se pueden descomponer sin importar el nivel. Los objetos Cíclicos y Esporádicos tienen un comportamiento similar dentro de los sistemas en tiempo real, en cuanto a que ambos contienen un único lazo de control que es planificable en el sistema. Por su parte los objetos Protegidos controlan el acceso a datos compartidos por uno o más hilos de ejecución, pueden proveer exclusión mutua. Un objeto Pasivo puede ser utilizado por otro u otros objetos si el modelo lo requiere.

Los objetos pasivos no tienen necesidad de sincronizarse con ningún otro objeto, porque su código es secuencial.

Ningún objeto puede utilizar operaciones de objetos bloqueados.

3.2.2 Reglas de Uso y Descomposición Para la utilización correcta de los modelos en HRT-HOOD, es necesario contar con algunas reglas e diseño, que indiquen los tipos de relaciones que se pueden establecer entre objetos.

En la tabla 1 se puede un resumen de las relaciones para el llamado de operaciones entre objetos.

Tabla 1. Relación de uso de operaciones entre objetos

		Objeto Invocado				
Objeto Invocador		**Cíclico**	**Esporádico**	**Protegido**	**Activo**	**Pasivo**
	Cíclico	Sí	Sí	Sí	Aser	Sí
	Esporádico	Sí	Sí	Sí	Aser	Sí
	Protegido	Asatc	Asatc	Sí	Aser	Sí
	Activo	Sí	Sí	Sí	Sí	Sí
	Pasivo	No	No	No	No	Sí

Esto quiere decir, que los objeto pasivos, por ejemplo, no puede invocar a ningún objeto diferente a otro pasivo.

Otra tabla que resulta útil para determinar las restricciones cuando se realizan estructuras jerárquicas se presenta en la tabla 2.

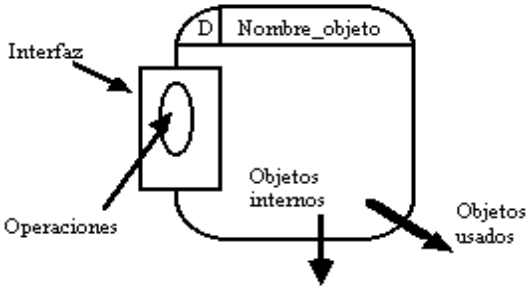
Tabla 2. Relación de descomposición

		Objeto Hijo				
Objeto Padre		**Cíclico**	**Esporádico**	**Protegido**	**Activo**	**Pasivo**
	Cíclico	Sí	Sí	Sí	No	Sí
	Esporádico	Sí	Sí	Sí	No	Sí
	Protegido	No	No	Sí	No	Sí
	Activo	Sí	Sí	Sí	Sí	Sí
	Pasivo	No	No	No	No	Sí

Un objeto cíclico, por ejemplo, puede contener en su interior cualquier tipo de objeto, con excepción de los objetos activos, o en otras palabras, cualquier objeto, excepto el activo, puede ser hijo de un objeto cíclico.

3.2.3 Representación de los objetos

Figura 17. Modelo de Objeto en HRT-HOOD



En figura 17, se observa la representación genérica de un Objeto en HRT-HOOD.

El nombre del objeto al igual que en UML, es su identificador, para que los diferentes objetos sean distinguidos unos de otros.

El objeto cuenta con una interfaz que define los servicios ofrecidos por el objeto a los objetos contenidos en el nivel donde se encuentra. Una de las características de los modelos orientado a objetos, es el ocultamiento de la información, y HRT-HOOD lo demuestra claramente en la interfaz. El objeto de HRT-HOOD cuenta con una interfaz que simplemente indica las operaciones o servicios que éste puede aportar dentro del nivel de jerarquía en el cual se encuentra ubicado; los detalles de su funcionamiento interno los deja ocultos, para que sean analizados en otro nivel jerárquico (si es posible).

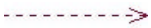
El identificador del tipo de objeto representado (D), es la única diferencia que tiene la representación del objeto en HRT-HOOD, con respecto a HOOD. Por medio de este identificador se logra diferenciar los tipos de objetos que se encuentran presentes en un modelo. La interpretación de los diferentes objetos se puede observar en la tabla 3.

Tabla 3. Identificador del tipos de Objetos

D	Tipo de objeto
P	Pasivo
A	Activo
C	Cíclico
S	Esporádico
Pr	Protegido

3.2.4 Relaciones Para interrelacionar los objetos de un nivel determinado, se utilizan algunas relaciones que de manera simple se definirán a continuación.

a. Relación de Asociación:  Permite relacionar los diferentes objetos, en función de las necesidades de utilización. Cuando hay una relación de este tipo entre dos objetos, normalmente se dice que el objeto origen de la flecha usa al objeto que se direcciona.

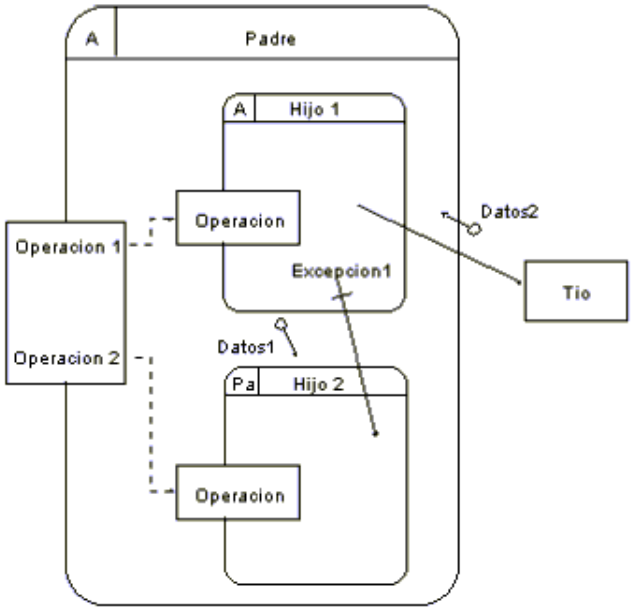
b. Implementación:  Indica cual de los objetos internos implementará las funciones del objeto Padre. Se representa como una flecha punteada dirigida desde la interfaz hasta el objeto que realiza la operación indicada.

c. Flujo de Datos:  Representa la transferencia de información desde un objeto hasta otro.

d. Flujo de Excepción:  Representa la información de la ocurrencia de una excepción en un objeto, alertando a otro para tomar realizar las acciones correspondientes. Por ejemplo, si el modelo utiliza un objeto que implementa la operación división, se puede utilizar un flujo de excepción, para anunciar a algún otro objeto la ocurrencia de la excepción presentada cuando aparece en un denominador el valor cero (0).

Un ejemplo general que recoge la presentación de las diferentes relaciones es ilustrado en la figura 18.

Figura 18. Ejemplo de modelo en HRT-HOOD.



La gráfica también muestra la relación jerárquica entre los diferentes objetos que interactúan en el modelo.

3.3 REDES DE PETRI

Con la aparición de las redes de Petri, se abrió un nuevo campo en el modelamiento de sistemas. Las redes de Petri permiten representar sistemas a cualquier escala de complejidad. Su fundamento matemático las ha convertido en una herramienta de modelamiento confiable para algunas personas, y un poco confuso para otras, pero actualmente están adquiriendo cada vez mayor importancia.

Por medio de las redes de Petri se pueden realizar descripciones (estructurales y funcionales) y validaciones formales de sistemas. Gráficamente es observable si un sistema presentará conflictos durante su funcionamiento. Dentro la etapa de diseño arquitectural pueden servir también, para realizar pruebas que en el prototipo final resultarían inadecuadas, bien sea por costos, tiempo, ubicación en la planta, etc.

Comparada con otros modelos de comportamiento dinámico gráficos, como los diagramas de las máquinas de estados finitos, las redes de Petri ofrecen una forma de expresar

procesos que requieren sincronía[8], además de que pueden representar fácilmente casos de secuencias, conflictos, concurrencia, etc.

Las redes de Petri modelan sistemas en base a dos de sus principales componentes: los lugares y las transiciones. Un sistema puede ser considerado como un conjunto de módulos que interactúa entre sí, los cuales pueden ser interpretados por sí mismos como subsistemas, para realizar análisis aislados (sin descuidar las relaciones con los demás componentes). El estado de los componentes del sistema es modelado en redes de Petri por medio de los lugares, y las acciones que hacen que el estado del sistema cambie, se representan con las transiciones.

El estado de un módulo depende de los eventos que han ocurrido anteriormente. Las acciones de un modulo pueden tener condiciones que restringen su uso, y crean una dependencia del estado del sistema en un instante de tiempo. En otras palabras, para que un evento ocurra se deben cumplir ciertas condiciones (Precondiciones del evento); después de que las acciones son realizadas, dichas condiciones pueden variar (PostCondiciones del evento).

Las acciones de los diferentes módulos del sistema pueden ejecutarse de manera paralela, para lo cual las redes de Petri, en comparación con otros métodos para modelar sistemas, presentan cierta ventaja.

3.3.1 Definición El poder y la gran acogida que han tenido las redes de Petri está en que tienen sus bases muy bien definidas, soportadas en definiciones matemáticas fuertes; esto garantiza que su aplicación modelará sistemas de manera correcta. La definición formal permite, además contar con:

-La seguridad de que no existirán ambigüedades (con un sistema bien modelado), y que el modelo será comprensible.

-Las bases necesarias para realizar validaciones formales.

-La posibilidad de la aparición fundamentada de extensiones de las redes de Petri existentes.

-Herramientas software para el diseño y simulación de los modelos implementados.

[8] CHAPARRO, David. Redes de Petri (Un editor Gráfico). Monterrey, 1993. Tesis (Maestría en Ciencias). Instituto Tecnológico y de Estudios Superiores de Monterrey.

Entre estos aspectos, cabe resaltar que el último es el que quizá tiene más inherencia con los modelos que se realizan, especialmente con los complejos, en vista de que vuelven *transparente* la definición formal de las redes de Petri. Por medio de las herramientas software, el diseñador no tiene la necesidad de preocuparse por el cumplimiento de las reglas establecidas en la definición, ya que estos se encargan de corregir o detectar errores estructurales, de sintaxis, semántica, etc., además de que permiten visualizar el comportamiento de la red conforme ocurren los eventos.

En este pequeño marco teórico no se presentará la definición formal de las redes de Petri, ya que esta requiere un análisis especial, que escapa a los objetivos de este proyecto, sin embargo se citan algunas bibliografías que contienen dicha teoría[9]. La definición aquí propuesta se basa en características de aplicación, dividiendo la representación en dos formatos, el modelo Estructural y del modelo Grafico de las redes de Petri.

- Representación Estructural de las Redes de Petri Las redes de Petri desde este punto de vista (aunque es general), son un formalismo operacional, que permite modelar sistemas a eventos discretos[10].

Este modelo clásico de una red de Petri (PN) es una 5-Tupla, $PN = (P, T, \alpha, \beta, M_0)$ donde:

$P = \{p_1, p_2, ..., p_m\}$ es un número finito de lugares representando estados estables del sistema modelado.

$T = \{t_1, t_2, ..., t_n\}$ es un número finito de transiciones y representan eventos. Estos eventos determinan la evolución de los estados en la Red de Petri.

$\alpha \subseteq (P \times T) \rightarrow \mathrm{N}$ es la función de incidencia previa
$\beta \subseteq (T \times P) \rightarrow \mathrm{N}$ es la función de incidencia posterior.

Estas dos funciones pueden representar acciones de transformación en las marcas. M_0 es el marcado o estado inicial del sistema.

[9] G. Rozenberg. Advances in Petri nets. New York, 1990
[10] GONZÁLES, Apolinar. Especificación de componentes mediante redes de Petri para el diseño y validación de sistemas de control de tiempo real. Valencia, 1999, 183 p. Tesis (Doctoral en ciencias de computación). Universidad Politécnica de Valencia. Departamento de informática de sistemas y computadores.

La cantidad de veces que un lugar *Pi* se repite a la salida o entrada del una transición *Tj*, se representa como $\#(Pi, \alpha(Tj)) = N$ o. De la misma forma se puede interpretar utilizando funciones de entrada o salida con lugares.

Los tokens, o marcas, son la parte dinámica de la red de Petri. Permiten saber o conocer el estado del sistema en determinado momento, como también su evolución el tiempo.

Una marca ***U*** de una red de Petri es una función ***U***: P $\rightarrow$ *N*. Es decir el lugar p_i tiene ***U***(p_i) tokens.

Ejemplo.

$\alpha(T1) = \{P1, P2\}$ Función de Entrada
$\beta(T1) = \{P3, P4\}$ Función de Salida
$\#(P1, \alpha(T1)) = 3$ Cantidad de repeticiones
$U(P1) = 4$ Cantidad de Tokens

Los anteriores ejemplos indican que la transición T1 tiene como lugares de entrada a P1 y P2, mientras que a su salida se encuentran los lugares P3 y P4. También se señala que el lugar P1 se repite tres veces a la entrada de T1. Finalmente se indica que el lugar P1 tiene cuatro tokens en el instante de tiempo analizado.
Nota: Las funciones de entrada y salida pueden ser utilizadas para los lugares.

Condición de activación de transiciones: La activación de una transición se logra cuando los lugares previos a esta, tienen la cantidad necesaria de tokens (o más) exigida por el número de repeticiones de cada uno de los lugares.

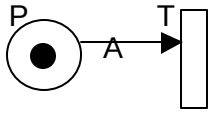
Una transición Ti en una red de Petri está habilitada, si para todo lugar Pi, $U(Pi) \geq \#(Pi, I(Ti))$.

Este tipo de representación se utiliza especialmente por aplicaciones que requieran un sentido matemático explícito, o para la implementación de programas de análisis de redes de Petri.

En de la representación estructural se puede analizar la evolución del sistema por medio de un análisis matricial, de la forma $M_{i+1} = M_i + Ce_t$, donde M_{i+1} es un vector que indica el marcado siguiente, M_i es el vector de marcado inicial, C es la matriz de cambio y e_t es el vector que indica cuales transiciones se encuentra habilitadas en el momento de análisis.

- Representación Grafica de las Redes de Petri En la interpretación gráfica de las redes de Petri, se representan los lugares como círculos u óvalos, y las transiciones como líneas o rectángulos. Además de estos dos elementos, aparecen los arcos, que se grafican como flechas, que conectan a los lugares con las transiciones en cualquier sentido (P→T, T→P). En otras palabras los gráficos se representan como G = (V,A), donde V son los lugares o las transiciones, y A los arcos. Los tokens se representan con pequeños puntos dentro de los lugares.

Ejemplo.



Aunque por lo general las redes de Petri se encuentran en forma gráfica se debe tener en cuenta que ambas representaciones son equivalentes.

- Diagramas de Alcanzabilidad Conocidos también como Espacios de Estados, o Árboles de Alcanzabiidad. Son diagramas que permiten analizar los diferentes marcados que pueden suceder durante la ejecución de un modelo. El estudio de estos diagramas proporciona un complemento en el análisis del modelado de alguna aplicación, en ellos se puede observar el comportamiento del sistema ante la ocurrencia de los diferentes eventos, se puede ver si determinado caso puede ocurrir, etc. Otro análisis importante que se puede realizar con los árboles de alcanzabilidad es la determinación de puntos de bloqueo del sistema, los cuales son marcados en los que ninguna transición se encuentra habilitada.

Los espacios de estados están compuestos por:

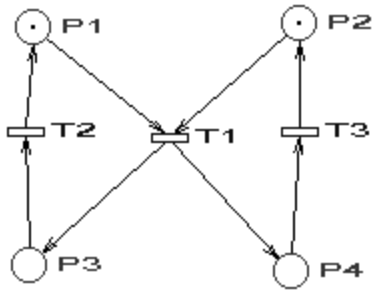
-Nodos: Cada nodo del diagrama representa un marcado diferente.

-Arco: Se genera uno para cada transición activada.

Ejemplo.

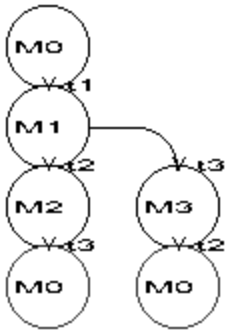
En la figura 19. se muestra una red de Petri, con un marcado inicial definido.

Figura 19. Ejemplo de Diagramas de Estados: Red de Petri



El árbol de alcanzabilidad correspondiente a este diagrama es el mostrado en la figura 20.

Figura 20. Ejemplo de Diagramas de Estados: Árbol de alcanzabilidad



En la figura 20, los nodos indican que existen cuatro tipos de marcados diferentes para los lugares existentes {P1, P2, P3, P4}:

M0 (Marcado Inicial) = {1,1,0,0}
M1 = {0,0,1,1}
M2 = {1,0,0,1}
M3 = {0,1,1,0}

También se puede observar por medio de los arcos, las transiciones que disparadas, cambian el estado del sistema, y estos apuntan hacia que estado se realiza el cambio.

En el ejemplo no se presentan bloqueos, ya que se puede ver que por cualquier camino de activaciones de transiciones se retorna al estado inicial (M0). Se dice entonces que el diagrama está *Vivo*.

3.3.2 Estrategia para modelamiento Si un sistema va a ser representado en redes de Petri sin tener modelos predecesores, resulta útil realizar una descripción de los estados y acciones con que cuenta el sistema, a manera de listado o tabla. Con los tipos de acciones definidas se mencionan las condiciones previas y posteriores a la ocurrencia del evento (igualmente como listado o tabla).

A continuación se presenta un ejemplo que muestra una de las maneras como un modelamiento puede ser realizado[11].

Ejemplo.

Un taller tiene tres máquinas: M1, M2 y M3; y dos operadores O1 y O2. El operador O1 puede trabajar las máquinas M1 y M2 y el operador O2 las máquinas M1 y M3. Las ordenes requieren de dos procesos, el primer proceso debe ser hecho por la máquina M1 y el segundo proceso puede ser hecho con la máquina M2 o la M3.

Los estados del sistema se muestran en la tabla 4 y las acciones en la tabla 5.

Tabla 4. Ejemplo de estrategia: Estados del sistema

Estados	
A	Una orden ha llegado y está esperando
B	Una orden ha sido trabajada y está esperando ser procesada por M2 o M3
C	La orden es completada
D	La máquina M1 está desocupada
E	La máquina M2 está desocupada
F	La máquina M3 está desocupada
G	El operador O1 está sin trabajo
H	El operador O2 está sin trabajo
I	El operador O1 está ocupando la máquina M1
J	El operador O2 está ocupando la máquina M1
K	El operador O1 está ocupando la máquina M2
L	El operador O2 está ocupando la máquina M3

[11] CHAPARRO, Op.cit.

Tabla 5. Ejemplo de estrategia: Eventos del sistema

Eventos	
1	Llega una orden
2	El operador O1 empieza la orden en M1
3	El operador O1 termina la orden en M1
4	El operador O2 empieza la orden en M1
5	El operador O2 termina la orden en M1
6	El operador O1 empieza la orden en M2
7	El operador O1 termina la orden en M2
8	El operador O2 empieza la orden en M3
9	El operador O2 termina la orden en M3
10	La orden es terminada y liberada

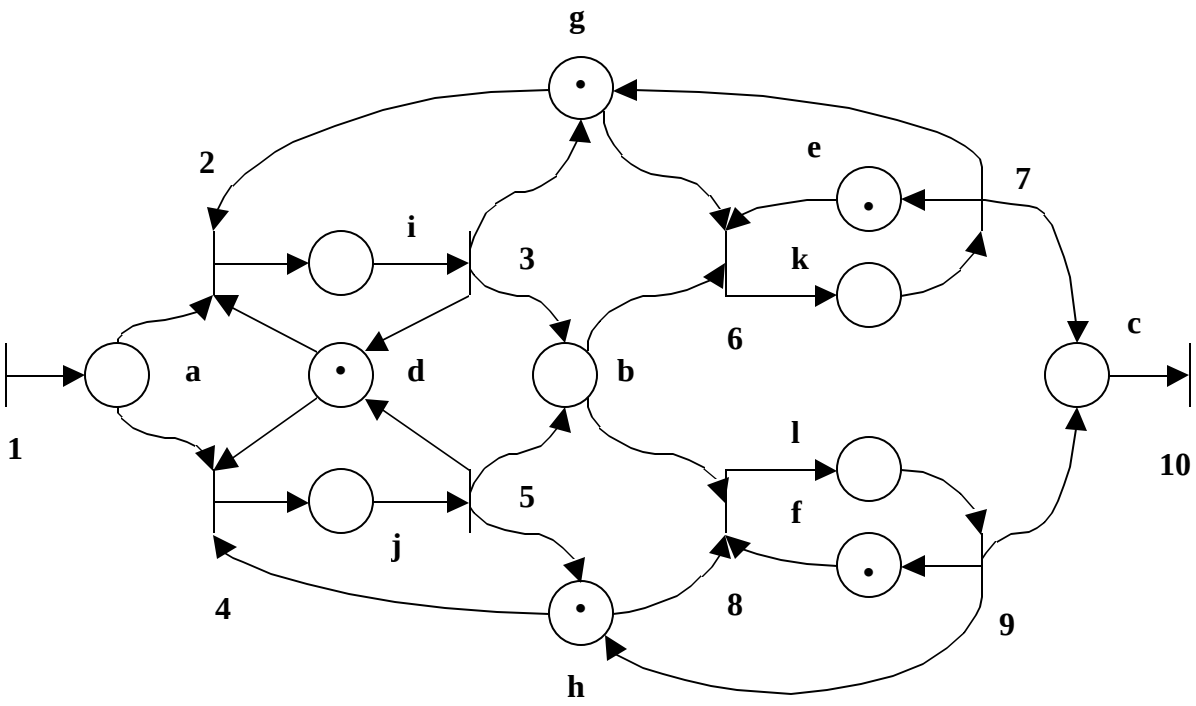
En la tabla 6 se ubican las relaciones condicionales previas y posteriores a las diferentes acciones, teniendo en cuenta que las condiciones iniciales son: D, E, F, G y H.

Tabla 6. Ejemplo de estrategia: Precondiciones y Postcondiciones

Precondiciones y Postcondiciones de cada evento		
Eventos	**Precondiciones**	**Postcondiciones**
1	Ninguna	A
2	A, G, D	I
3	I	G, D, B
4	A, H, D	J
5	J	B, H, D
6	B, G, E	K
7	K	C, G, E
8	B, f , H	I
9	L	C, F, H
10	C	Ninguna

La red de Petri correspondiente a las definiciones hechas en este ejemplo, se muestra en a figura 21.

Figura 21. Ejemplo de Estrategia de diseño: Red de Petri



3.3.3 Evolución de las redes de Petri Las redes de Petri fueron publicadas originalmente en la tesis doctoral *Kommunikation mit automaten* (Comunicaciones con Autómatas), por C*arl Adam Petri* (Alemán), en 1962. Las primeras redes fueron llamadas Condition/Event Nets (CE-nets). En este tipo de redes, se permitía que los lugares guardarán solo una marca, con lo que se representan modelos en lógica booleana (verdadero y falso). La contribución de muchas personas llevó a la aparición de las Place/Transition Nets (PT-nets); estas redes le permiten a un lugar contener varias marcas, logrando aumentar el alcance de las CE-nets hasta la manipulación de variables de tipo entero. La necesidad de complementar matemáticamente las redes de Petri, excluyeron para consideraciones teóricas a las PT-nets, orientándose más la investigación hacia las CE-nets; resultado de tal inclinación fueron las Elementary Nets (EN-nets), que guardaban una muy estrecha relación con las CE-nets, pero para ese tiempo solucionaron algunos de los problemas presentados por estas últimas.

Para aplicaciones prácticas las PT-nets fueron ampliamente utilizadas, sin embargo el nivel tan bajo en que se encontraban, obligaba a los desarrolladores y analistas a incorporar sus propias modificaciones, agregando niveles de prioridad, tiempo en las transiciones, utilizando variables globales, etc. Así aparecieron muchos tipos de redes de Petri, sin embargo, la mayoría de ellas carecían de fundamentos matemáticos que las soportarán, razón por la cual no lograban ser generales, sino que por el contrario, resultaban ser redes confusas y demasiado especificas al ambiente sobre el cual eran desarrolladas.

El gran avance se logró con la aparición de las Predicate/Transition Nets (PrT-nets). Estas fueron creadas sin tener ningún proceso específico en cuenta. Pertenecen a la primera generación de redes de Petri de alto nivel (hl-nets).

Pronto se encontraron problemas y empezaron a salir otros modelos que terminaron en plantear las High-Level Petri Nets (HL-nets). Para evitar confusiones con las hl-nets, se determinó nombrarlas como Coloured Petri Nets(CP^{87}-nets). Antes de estas redes aparecieron otros tipos de redes coloreadas, por eso estas se diferencian con el número 87 (año de su aparición), aunque comúnmente se conocen como CP-nets.

Un resumen de la evolución histórica, mencionando las redes más destacadas, se puede observar a continuación:

Primer Nivel
CE-Nets (Condition/Event Nets)
EN (Elementary Nets)

Segundo Nivel
PT-Nets (Place/Transition Nets)

Tercer Nivel
HL+ADT (High Level Petri Nets with Abstract Data Types)
Algebraic Petri Nets (ALG)
*OBJSA-Nets
ER-Nets (Environment Relationship Nets)
Prod-Nets (Product Nets)
WN-Nets (Well Formed Nets)
*RN-Nets (Regular Nets)
Traditional High-Level Petri Nets
*PrT-Nets (Predicate/Transition Nets)
*CP-Nets (Coloured Petri Nets - CPN)

Con algunos tipos de redes, actualmente se puede lograr pasar a la implementación de una manera muy práctica, ya que especifican gran parte de la información en forma de lenguaje de programación; es el caso de las redes de Petri de alto nivel (HLPN, High Level Petri Net). Las redes de Petri Coloreadas (CPN, Coloured Petri Net), son un ejemplo claro de redes de Petri de alto nivel, donde las marcas (Tokens) pueden transportar información, los lugares (Places) pueden aceptar solo marcas de determinada clase, las transiciones (Transitions) pueden estar condicionadas, y otra cantidad de características que permiten implementar de manera sencilla muchas funciones, logrando obtener de esta manera modelos menos complejos y más útiles.

3.3.4 Redes de Petri Coloreadas (CPNs) Las redes de Petri Coloreadas son un lenguaje de modelamiento desarrollado para modelar sistemas en los cuales la comunicación, la sincronización y la compartición de recursos tienen gran importancia. Combina las propiedades de las redes de Petri Ordinarias con principios de lenguajes de programación. Las redes de Petri proveen las primitivas para la interacción del proceso, mientras que los lenguajes de programación, aportan las primitivas para la definición de tipos de datos y la manipulación de ellos.

La relación entre las redes de Petri Coloreadas y las Ordinarias es análoga a la relación existente entre los lenguajes de programación de alto nivel y el código assembler. En teoría tienen el mismo poder computacional, pero en la práctica los lenguajes de alto nivel son mucho más poderosos al momento de programar, porque tienen una mejor estructura, módulos y tipos.

Algunas de las principales características de las redes de Petri se mencionan a continuación:

- Manejo de información en las marcas: Los tokens no solo pueden tener relaciones de presencia y ausencia en un lugar, en este tipo de redes los tokens tienen atributos que les permiten transportar información dentro de la red. Por ejemplo, los tokens pueden llevar datos temporales, de temperaturas, de sincronización, etc.

- Tienen la posibilidad de implementar estructuras jerárquicas: Es una característica muy importante de estas redes, ya que por medio de estas se logra reducir la complejidad en el análisis de sistemas robustos.

- Muy estrecha relación con los lenguajes de programación: La implementación de ciertas aplicaciones complejas, se ve reducida, al contar con propiedades de lenguajes de programación muy útiles para tales casos.

- Modelamiento de sistemas concurrentes: Al igual que las redes de Petri ordinarias, las Coloreadas permiten que varios componentes de un sistema ejecuten sus operaciones al mismo tiempo. Esta característica es muy útil si lo que se desea modelar tiene implicaciones temporales estrictas.

- Representación gráfica: La forma gráfica es intuitiva, muy sencilla de entender, debido a las propiedades jerárquicas de CPN y a la utilización de código simple.

- Semántica bien definida: Permite representar diagramas que pueden ser interpretados en un solo sentido.

- Son de uso general: Sus aplicaciones son extendidas. Permiten modelar sistemas software, hardware, financieros, militares, etc.

- Pocas pero poderosas primitivas: Las pocas primitivas con que cuentan las redes de Petri Coloreadas no son impedimento para realizar cualquier tipo de aplicación. Por el contrario, permiten que el diseñador con conceptos básicos de programación, logre asimilar la utilización de este tipo de redes muy fácilmente.

- Permiten relacionar fácilmente los estados con las acciones: Muchos lenguajes de descripción, analizan por independiente a los estados y las acciones del sistema.; esto puede crear confusión para sistemas grandes. Las CPNs integran la descripción de las acciones con los estados, obteniendo modelos muy prácticos.

- Modelan funciones temporales: Enmarcadas en un ambiente estocástico, las redes de Petri coloreadas, manejan los conceptos temporales en representaciones simples, logrando incluir diferentes formas de aplicación: no solo las transiciones pueden incluir retardos, los arcos tienen la misma posibilidad. Si es necesario, se pueden implementar modelos que no manejen aspectos temporales.

- Herramientas de modelamiento: Permiten realizar validaciones de forma dinámica y un poco alejadas de la formalidad propia de la definición de las CPNs. Debido a su semántica tan clara, los diseñadores de software han presentado una diversidad de simuladores de redes de Petri Coloreadas, que permiten facilitar la labor de diseño.

Entre las propiedades estructurales y los componentes de las redes de Petri coloreadas se encuentran:

- Lugares, conjunto de colores (colour sets), marcas (tokens), marcados, marcado inicial.

- Transiciones, guardas, arcos, expresiones de arco.

- Elementos de enlace, reglas de enlace, reglas de ocurrencia.

- Pasos, secuencias finitas e infinitas de ocurrencias.

- Sustituciones de transiciones y lugares de conexión (socket places).

- Subpáginas, lugares puerto, instancias de pagina.

- Fusión de lugares, conjuntos de fusiones globales, de pagina y de instancias.

- Pagina de Jerarquía.

Por medio de estos de elementos, es posible realizar el modelamiento de sistemas, tanto a nivel jerárquico como no jerárquicos, con CPNs, implementando todas las características que este *lenguaje* de diseño entrega.

3.3.5 Campos de aplicación de las redes de Petri La aplicación de las redes de Petri se está extendiendo a nivel a mundial hacia muchos campos, debido a la flexibilidad, y carácter general que éstas tienen. Países como Austria, Dinamarca, Estados Unidos, España, Francia, Alemania, Rusia, Suecia, Dinamarca, Japón y Colombia, entre muchos otros, y empresas como Boeing, Hewlett Packard, Samsung, Siemens, Mitsubishi, etc, se encuentran utilizando las Redes de Petri, para validar sus procesos, proyectos y desarrollos tecnológicos.

Algunas de las aplicaciones que pueden tener las redes de Petri se mencionan enseguida.

- Planificabilidad en tiempo real Para analizar si los sistemas diseñados son planificables en tiempo real, es decir, si sus tareas se cumplen dentro de los plazos establecidos, se pueden utilizar distintos métodos numéricos, como por ejemplo Rate monotonic y Deadline monotonic. Dichos algoritmos matemáticos están condicionados a que las tareas sean cíclicas e independientes entre sí (entre las características más primordiales).

La segunda condición planteada, limita el estudio del sistema a validar por medio de los métodos mencionados, debido a que por lo general existen relaciones entre objetos, como en el caso de las redes de comunicaciones con respecto a los programas receptores, barreras de sincronización, etc., en los que se relacionan las tareas entre sí. Por este motivo, se aprovechan las ventajas de las redes de Petri, con el fin de observar el funcionamiento temporal del sistema.

- Sistemas militares Muchos de los desarrollos tecnológicos han aparecido durante las guerras. Esta industria requiere de estrategias de modelamiento que permitan de manera eficiente la implementación de ciertos proyectos. Algunas aplicaciones realizadas para los militares son las siguientes:

  - Compuertas de Comunicaciones Militares en Australia
  - Redes de Influencia para fuerza aérea de Estados Unidos
  - Simulador de mísiles en Australia
  - Control y comando de sistemas navales en Canadá

- Diseño de protocolos y redes Muchos de los protocolos implementados en la actualidad han sido modelados en sus fases de diseño; las redes de Petri, en especial las Coloreadas permiten la implementación de estos desarrollos. Se pueden mencionar algunas de las aplicaciones hasta donde ha llegado el alcance de las redes de Petri:

  - Redes Inteligentes en Alemania Telekom
  - Configuración de control para IEEE 802.6 en Telstra Research Labs
  - Establecimiento de políticas en el Protocolo de Bus de Campo en Japón
  - Servicios RDSI en Telstra Research Laboratories
  - Protocolo para sistemas de Audio/Video en Bang & Olufsen
  - Protocolos TCP en Hewlett-Packard
  - Redes de Área Local en la Universidad de Las Palmas
  - Algoritmos UCP en redes ATM en la Universidad de Aarhus Dinamarca
  - Protocolo BRI in redes RDSI
  - Sistemas de Gestión de Redes en RC International A/S
  - Comunicación de Interprocesos en Pool IDA en King's College

- Otros campos Las redes de Petri han sido ampliamente utilizadas en otros sistemas como desarrollos de Software y Hardware, control de procesos, programas nucleares, sistemas económicos, etc. Con esto se demuestra la importancia que están adquiriendo en los esquemas de modelamiento de sistemas.

3.3.6 Herramientas software para el modelamiento de redes de Petri Para el modelamiento de las redes de Petri hay distintas herramientas, que cuentan con características propias. Algunas son más dinámicas que otras, otras entregan más servicios, algunas no modelan redes estocásticas, etc. Algunas de las herramientas disponibles se presentan a continuación:
Modelamiento de diagramas temporizados, evolución dinámica del sistema:

- Visual Object Net ++
- HPSim
- PESIM

Menor dinámica en el modelamiento. Presenta diagramas de Alcanzabilidad:

- PESIM
- CABERNET

Redes de Petri con Relación de Ambiente (ER-Nets):

- CABERNET

Modelamiento de redes de Petri Coloreadas (CPN)[12]:

- Alpha
- ANARCO
- CPN/AMI
- DESIGN/CPN
- DESIGN/OA
- EXSPECT
- FORSEE
- INA
- LOOPN
- PAPETRI
- POSES
- PROMPT
- PNTBLSIM
- TemPRO
- VisualSIMNET
- VOLTAIRE

[12] Universidad de Aarhus [En línea]. [Dinamarca], marzo de 2003 [Citada en febrero de 2002]. Disponible en internet <http://www.daimi.au.dk/Cpnets> .

4. SISTEMAS DE CONTROL DISTRIBUIDOS

En la actualidad industrial existe la tendencia de implementar sistemas de control que están basados en los sistemas distribuidos[13]. Esto es interrumpir el lazo de control, y adicionar redes de computadores para aplicaciones distribuidas. La funcionalidad es incrementada notablemente debido a la posibilidad de intercambiar información entre el sistema de control, la red y el software. Los tipos de redes ampliamente difundidos para estas aplicaciones son los buses de campo como DeviceNet, pero también se pueden realizar aplicaciones que incluyan otros tipos de redes como Ethernet y FireWire, entre otras[14].

Los sistemas de computadores, por lo general, se componen de redes que conectan los diferentes nodos del sistema, donde el acceso a la red de cada nodo se realiza a través de compuertas. Un nodo en un sistema de control industrial puede incluir: sensores, interfaces para los actuadores, microcontroladores, interfaces para acceder a la red, etc.

En este capítulo se tratará un poco la descripción los sistemas de control en redes distribuidas, enfocadas a la aplicación de conceptos de tiempo real. Además se plantearán algunas definiciones para los sistemas de control, distribuidos y de tiempo real.

4.1 SISTEMAS CONTROLADOS EN RED

Un sistema de control cerrado en el cual su lazo de retroalimentación pasa a través de una red de comunicaciones, la cual puede estar compartida con nodos que no pertenezcan al sistema de control, se conoce como Sistema Controlado por Red (NCS). En otras palabras, los NCS describen una arquitectura donde la comunicación de al menos uno de los lazos que relacionan al controlador y la planta, reside sobre una red.

Los NCS son especialmente utilizados cuando los componentes del sistema de control se encuentran dispersos geográficamente.

---

[13] TÖRNGREN, Martin; ELKHOURY, Jad y SANFRIDSON, Martin y REDELL, Ola. Modelling and Simulation of Embedded Computer Control Systems: Problem Formulation. Mechatronics Lab. Estocolmo, Suecia. [sf]. 128 p.

[14] MARTÍNEZ, diego. Planteamiento de un estilo de diseño formal para el desarrollo de Sistemas de Control Distribuidos. En : SEMINARIO INTERNACIONAL AVANCES Y TENDENCIAS EN ELECTRÓNICA ALTO NIVEL. Corporación Universitaria Autónoma de Occidente. Santiago de Cali. Agosto 14 a 16 de 2000.

La introducción de estos sistemas, trae consigo la obtención de ventajas, tales como, la utilización óptima de los recursos, fácil mantenimiento, bajos costos de instalación, uso adecuado de espacios, incremento en la cobertura, posible realización de análisis (estabilidad, rendimiento, tiempos de operación), etc.

Algunas características que justifican el uso de los NCS, son las siguientes:

- Los componentes del sistema, como por ejemplo los sensores y los controladores, pueden estar espacialmente distribuidos.

- Flujo de diferentes tipos de datos en la red

    - Datos críticos
      -Variables periódicas. Por ejemplo, datos de muestreo para actualizar la salida del controlador
      -Variables aperiódicas. Como por ejemplo la señal de activación de una alarma
    - Datos no críticos
      -Inicialización de nodos o sistemas, información de instalación
      -Diagnostico y supervisión de la red
      -Interacción con sistemas de alto nivel

- Las redes de comunicación utilizadas cumplen ciertas condiciones

    - Altas velocidades de transmisión, característica propia de los protocolos de bus de campo, en los cuales el arbitramiento y la detección de errores tiene importancia particular
    - Capacidad de transferencia de datos críticos y no críticos temporalmente
    - Confiabilidad alta en la transmisión
    - Operabilidad bien definida para el ambiente de aplicación
    - Facilidad de instalación y mantenimiento
    - Económicas

4.2 SISTEMAS DISTRIBUIDOS

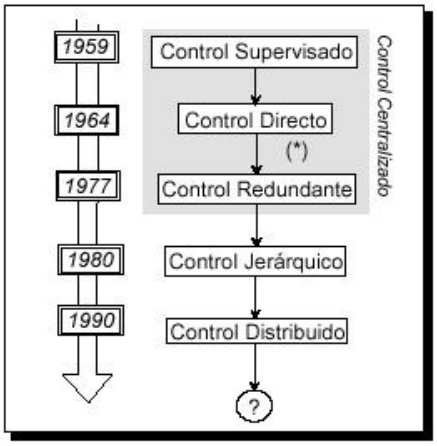
La comunicación de los datos en una red, se puede realizar de dos maneras: Centralizada y Distribuida. La diferencia esencial entre estos dos sistemas de computación, radica en la ubicación y control de la información. En los sistemas centralizados la información se almacena en un solo nodo, y es entregada a los demás nodos por medio de una red de comunicaciones. En un sistema distribuido, la información se encuentra en diferentes puntos del sistema que asociados a la red.

4.2.1 Evolución de los Sistemas de Computación en redes Los sistemas centralizados fueron la única opción de computación existente hasta el año de 1985, cuando aparecieron los sistemas distribuidos.

La aparición de los sistemas distribuidos fue motivada por el progreso de las arquitecturas de los microprocesadores (8, 16, 32 y 64 bits) y su disminución en costos, y por desarrollo de Redes de Área Local (LAN) de alta velocidad, las cuales permitían conectar muchos computadores con velocidades de transferencia de millones de bits por segundo (mb/s).

La figura 6.1, ilustra la evolución histórica que trajo como resultado, la implementación de los sistemas distribuidos aplicados al control de procesos.

Figura 22. Evolución de los sistemas de control



Control Supervisado:

-Un único ordenador controla los parámetros de todos los dispositivos, con posible intervención humana.
-Bajo costo.
-Diferentes requisitos temporales (complejos).
-Sensibilidad a fallos (3 a 6 meses en la época)

Control Directo:

-Un ordenador implementa directamente la ley de control.
-Implementación de control muy sofisticado (control adaptativo, óptimo, multivariable, etc.)
-Limitado a control de pocos procesos.

Control Redundante (Dual):

-Dos ordenadores implementan la ley de control. Un intercambiador activa uno u otro en caso de falla.
-Duplicación de cableado.
-Duplicación de software.
-El intercambiador es crítico.
-Incremento de la complejidad del diseño.

Control Jerárquico:

-Varios ordenadores se distribuyen las tareas de control en diversas capas organizadas según los requisitos temporales de los procesos.
-El diseño se facilita al dividir distintas tareas por sus requisitos temporales.
-Rigidez en la asignación de tareas a ordenadores.
-Diversidad en los dispositivos de control (mayor costo).

Control Distribuido (DCS):

-Ordenadores similares entre los que se dividen todas las tareas de control. Pueden tener características jerárquicas (modelos híbridos).
-El soporte de fallas es mucho mayor.
-Necesidad de alto flujo de datos en caso de distribución automática de tareas.
-Mayor eficiencia debida al proceso paralelo.
-Menor rigidez en los dispositivos.
-Mayor escalabilidad.
-Cobertura de plantas de mayor tamaño.

4.2.2 Sistemas centralizados Vs Sistemas Distribuidos Los sistemas centralizados se caracterizan por la existencia de un nodo que se encarga de administrar toda la información del sistema. Aunque es una opción bastante simple para ser implementada, si se compara con los sistemas distribuidos, presenta ciertas desventajas; algunas de estas desventajas son las siguientes:

- Económicamente En los sistemas centralizados se requiere que el nodo principal tenga una capacidad de procesamiento alta, para lograr atender todas las peticiones que le pueden realizar. Esto implica que el *servidor* debe estar ubicado en equipos que tengan características especiales, en velocidad de procesamiento, capacidad de almacenamiento, etc. La ley de Grosh formula que *el poder de computo de una CPU es proporcional al cuadrado de su precio*, de esta forma, si se paga el doble se obtiene el cuádruple del desempeño. Por su parte los sistemas distribuidos no requieren de

unidades de procesamiento poderosas (dependiendo de la aplicación), ya que la información no se almacena en un solo punto, sino que se encuentra repartida en los diferentes componentes de la red.

- Confiabilidad La posibilidad de recuperación ante fallas que ofrece el sistema distribuido es una de las principales características de esta arquitectura. En un sistema centralizado, si se daña el nodo principal, generalmente la red pierde el control; de otro lado, en una red distribuida la carga se encuentra dividida, de tal manera que el sistema no colapsará por la ausencia de alguno de sus nodos.

- Crecimiento Con los sistemas distribuidos se puede realizar la adición de nuevos nodos o unidades de computo, sin mayores inconvenientes.

- Velocidad de acceso a los datos El acceso a la información es realizada de una manera más rápida, ya que los nodos pueden intercomunicarse entre ellos sin necesidad de tener que acceder a bases de datos u otros elementos centrales, las cuales, cuando la información es requerida pueden estar ocupadas.

4.2.3 Definición Una red es un conjunto de dispositivos interconectados, que pueden intercambiar datos (voz, imagen, etc)[15]. Las aplicaciones de red son sistemas distribuidos implementados sobre una red. Una de las aplicaciones de red más comunes son las NCSs.

Un sistema distribuido es un conjunto de entidades que se comunican entre ellos a través de mensajes, los cuales son enviados sobre vías de comunicación. (Entidades: procesos, computadoras, redes computadoras dispositivos, procesadores etc.). Otra definición dice que un sistema distribuido es aquel donde dos o más maquinas colaboran para la obtención de un resultado. En todo sistema distribuido se establecen una o varias comunicaciones siguiendo un protocolo preestablecido mediante un esquema cliente servidor.

Algunas características de los sistemas distribuidos son las siguientes:

- Uso de un sistema de comunicación.
- Ausencia de memoria común.
- Sincronización del trabajo.
- Paralelismo en el control de los procesos
- Tiempo real en el control y las comunicaciones
- Apertura (Uso de estándares)

[15] GÓMEZ, Roberto. Introducción a los Sistemas Distribuidos. Monterrey, México. Instituto Tecnológico y de Estudios Superiores de Monterrey. Departamento de Ciencias Computacionales. 1996.

- Flexibilidad de adaptación a distintas tareas
- Escalabilidad
- Comunicación a través de mensajes.

A pesar de la gran cantidad de utilidades con que cuentan los sistemas distribuidos, tienen también desventajas como:

- El software utilizado en cada nodo es más complejo que con los sistemas centralizados.
- Dificultad en el reconocimiento de la red por parte de los usuarios.
- Incomprensión en la distribución de las tareas entre el operario y el sistema.
- Posibles problemas en la red: Pérdidas de mensajes, saturación en el tráfico, expansión, etc.
- Problemas de seguridad, razón por la cual los diseños del software deben considerar complejos sistemas para resolver este inconveniente

Los campos donde se concentra la investigación en sistemas distribuidos son:

- Redes de área local y extensa
- Sistemas operativos distribuidos
- Bases de datos distribuidas
- Servidores de ficheros distribuidos
- Lenguajes de programación concurrentes y distribuidos
- Lenguajes de especificación para sistemas concurrentes
- Teoría de algoritmos paralelos
- Teoría de computación distribuida
- Arquitecturas paralelas y estructuras de interconexión
- Sistemas ultraconfiables y tolerantes a fallos
- Sistemas distribuidos de tiempo real
- Técnicas de resolución cooperativa de problemas
- Depuración distribuida
- Simulación distribuida
- Aplicaciones distribuidas
- Metodologías para el diseño, construcción y mantenimiento de sistemas distribuidos grandes y complejos, y herramientas de trabajo cooperativo.

4.3 SISTEMAS DE CONTROL

El objetivo de los sistemas de control es gobernar y predecir el comportamiento de procesos. Dicho comportamiento se rige a partir de parámetros establecidos como *leyes de control*. Existen diferentes tipos de sistemas de control dependiendo del proceso se maneja en el sistema: sistemas neumáticos, hidráulicos y electrónicos, entre otros.

En general, un sistema de control es aquel que maneja un conjunto de variables de salida de un proceso físico y modifica las variables de entrada para corregir o limitar las desviaciones medidas respecto a los valores deseados.

Definiendo un *sistema* como un objeto en el que variables de distintos tipos interactúan y producen señales observables, se pueden distinguir cuatro tipos de variables.

-Variables de Entrada
-Variables de Perturbación
-Variables de Salida
-Variables de Estado

Los sistemas de control forman parte, actualmente, de procesos complejos como los aeronáuticos, los de centrales nucleares, diseño de robots y son un componente básico en los procesos industriales y de fabricación modernos.

Tradicionalmente, los sistemas de control se realizaban con tecnología analógica. A finales de los años cincuenta se dieron los primeros casos de empleo de computadores digitales. Inicialmente eran sistemas de supervisión que se empleaban para la determinación de los parámetros óptimos de los controladores analógicos. A partir del año 1962 se comienzan a emplear ordenadores para la realización de funciones de control digital directo. La creciente potencia de los ordenadores motivó a que se usaran para realizar simultáneamente labores de control y supervisión de la planta, lo que complicó progresivamente el desarrollo del software. Esta tendencia ha mantenido, y actualmente la gran flexibilidad y el incremento continuo de la potencia de los sistemas informáticos han permitido incluir un elevado número de funciones en un sólo computador y motivar el desarrollo de métodos de control muy sofisticados

Los sistemas de control se pueden clasificar de manera funcional y de manera estructural. Funcionalmente los tipos de sistemas que se pueden encontrar son los Continuos y Discretos. Estructuralmente o relacionados con su arquitectura de implementación, se clasifican en Sistemas de Lazo Abierto y de Lazo Cerrado.

4.3.1 Sistemas continuos de control Son sistemas que varían permanentemente en el tiempo. En general, estos sistemas se representan en términos de ecuaciones diferenciales lineales o no lineales. En muchos casos las ecuaciones no lineales se pueden linealizar bajo ciertas suposiciones.

El análisis y diseño de este tipo de sistemas de control es el objeto de la *teoría clásica de control*. Estas técnicas se han diseñado con la suposición de que los controladores serían también sistemas continuos. La realización con computadores de estos elementos

presenta problemas específicos. El computador recibe las señales de salida del proceso y actúa sobre las entradas en instantes de tiempo discretos. Por tanto, se debe describir el cambio de las señales de instante de muestreo en instante de muestreo y sin información sobre el comportamiento del sistema entre tanto.

Los modelos matemáticos que se emplean suelen suponer que hay una sincronización entre el instante de muestreo y el accionamiento correspondiente al muestreo previo. Si esta condición no se cumple, el funcionamiento del controlador no será el deseado y, por lo tanto, el comportamiento del proceso controlado tampoco lo será. En consecuencia, cuando estos controladores se realizan por computador, se debe asegurar que la respuesta del programa ocurrirá de acuerdo con esta hipótesis.

4.3.2 Sistemas discretos de control Los sistemas discretos son aquellos que sólo cambian en instantes discretos de tiempo y como respuesta a un evento externo. Por esta razón, a estos sistemas también se les conoce como sistemas de eventos discretos. Los métodos de control de estos sistemas son radicalmente diferentes a los de sistemas continuos. Se suelen realizar mediante métodos basados en representar de alguna forma el estado del sistema y en especificar los cambios de estado en función de las variaciones en las entradas.

La utilización de técnicas informáticas para realizar estos tipos de sistemas es aún mayor que en los sistemas continuos de control. La realización práctica de este tipo de sistemas también requiere una respuesta del sistema en un intervalo de tiempo que comienza cuando se produce el evento que la activa. En caso de no cumplirse estos requisitos, el sistema no puede funcionar adecuadamente.

Algunos controladores forman parte de sistemas de seguridad crítica. Estos son sistemas en los que una falla puede tener fuertes consecuencias (económicas o humanas inclusive). Un ejemplo son los sistemas de control por computador forman una parte vital de los modernos sistemas de aviónica. Las consecuencias que una falla podría provocar son fácilmente imaginables.

4.3.3 Sistemas de control en lazo abierto y en lazo cerrado Los sistemas de control en Lazo Cerrado, se refieren a aquellos que actúan sobre la planta o el proceso, sin considerar el valor de la señal de salida, es decir, la salida no se compara con la entrada. Hay una gran variedad de ejemplos de esta forma de trabajar. Uno de ellos es la lavadora: las prendas son depositadas en estas maquinas, después de realizar el proceso de limpieza por un determinado tiempo, la lavadora detiene su operación, pero no garantiza que la ropa haya quedado limpia.

Estos sistemas de lazo abierto se basan en temporizaciones y en un conocimiento exacto entre la entrada y la salida, sin ninguna consideración sobre las posibles perturbaciones

internas o externas. El ruido o perturbación sobre estos equipos de control provocarán que no se realice la tarea deseada.

En los sistemas de control en cadena cerrada, la señal de salida se compara con la entrada, para obtener una señal de error, la cual actúa sobre la planta. Debido a esta característica estos sistemas son más inmunes a las perturbaciones. Ejemplo de este tipo de sistemas es una plancha, a la cual se le indica la temperatura deseada, y por medio de un termostato regula la temperatura real de la resistencia.

Los sistemas de control de lazo cerrado o retroalimentados tienden a ser inestables, mientras en cadena abierta resulta más fácil de desarrollar por que la estabilidad no está tan comprometida. El concepto de inestabilidad o de estabilidad está relacionado con la capacidad de que un sistema de control sea capaz de seguir la señal de entrada sin perder el gobierno de éste. Un ejemplo inmediato, los sistemas de control de una aeronave deben de garantizar la estabilidad de ésta ante las órdenes dadas por el piloto y con independencias de las perturbaciones que sufra el avión.

4.4 SISTEMAS EN TIEMPO REAL

El tiempo real en el análisis de estrategias de control permite conocer los momentos exactos en que los fenómenos deben ocurrir, antes de que el sistema degrade su desempeño o llegue a puntos críticos de inestabilidad. Se debe aclarar que un sistema de tiempo real debe ser más que rápido, predecible; aunque una característica de estos sistemas debe ser la rapidez y la eficiencia, éstas no definen un sistema de tiempo real sino existe un análisis de los tiempos de respuesta del sistema.

La importancia de estos sistemas parte de la necesidad de tener un dominio completo sobre el proceso a controlar. Se presentan casos en los que no es suficiente con que el sistema funcione perfectamente si no se cumple con ciertas condiciones temporales. Esto es particularmente importante en los NCS. Estos procesos calculan una acción de control dependiendo del valor de la señal de salida, es importante conocer y controlar los tiempos en los que el valor de la señal de salida pasa a través del lazo de retroalimentación, conocer el tiempo que se tarda el calculo del error, etc., para implementarlos en sistemas que tengan ciertos riesgos, humanos o económicos.

Un sistema en tiempo real puede ser definido como:

Cualquier sistema que tiene que responder a estímulos generados externamente dentro de un plazo especificado y finito.

Cualquier sistema en el que el instante en que se produce la salida es significativo. Esto se debe habitualmente a que la entrada corresponde a algún movimiento en el mundo físico, y la salida está relacionada con este movimiento. El intervalo entre el instante de la entrada y el de salida debe ser suficientemente pequeño para que sea oportuno.

En muchos campos se ha notado la necesidad de implementar métodos específicos para los sistemas de tiempo real:

- Especificación y verificación. El reto fundamental en la especificación y verificación de sistemas de tiempo real es cómo incorporar el tiempo en la descripción del sistema. Es necesario desarrollar métodos que permitan describir las restricciones de tiempo del sistema y que sirvan para analizar si se cumplen. Algunas de las estrategias desarrolladas para este fin son las redes de Petri y HRT-HOOD (que ha sido diseñado pensando en el modelamiento de este tipo de sistemas).

- Teoría de planificación. El objetivo principal de la teoría de planificación es ordenar la ejecución de las actividades de un sistema, de forma que se cumplan los requisitos temporales. Se deben desarrollar métodos que permitan comprender, predecir y facilitar el mantenimiento del comportamiento temporal del sistema.

- Sistemas operativos. Deben proporcionar un alto nivel de abstracción a los programadores y, a la vez, cumplir las restricciones temporales, que dependen de detalles de bajo nivel de la implantación y del entorno. El tiempo y los principios de planificación deben ser elementos centrales en el diseño de aspectos del sistema operativo, como los métodos de gestión de recursos, las tareas de aplicación, etc.

- Lenguajes de programación. El aumento de la complejidad de los sistemas de tiempo real demanda abstracciones de alto nivel en los lenguajes de programación. Algunas características deseables son mecanismos para gestión del tiempo, análisis de planificabilidad, módulos reutilizables de tiempo real y mecanismos para desarrollar programas distribuidos y tolerantes a fallos.

- Metodologías de diseño. Son necesarias metodologías de diseño que incorporen y permitan razonar sobre el tiempo desde las primeras fases de desarrollo.

- Bases de datos distribuidas. Los datos en los sistemas de tiempo real sólo son válidos durante un cierto período de tiempo. Para satisfacer las restricciones temporales, se debe aumentar el paralelismo de las transacciones y se deben integrar los algoritmos de control de acceso a la base de datos y mecanismos de planificación.

- Inteligencia artificial. La utilización de métodos, basados en sistemas de tiempo real, pasa por controlar el tiempo de cómputo de su aplicación. Se debe proporcionar la mejor respuesta posible en un plazo de tiempo que puede ser variable.

- Tolerancia a fallos. Los sistemas de tiempo real se emplean en sistemas que requieren un nivel alto de confiabilidad. En este sentido, son necesarios métodos generales de detección de errores, localización y recuperación de fallas, reconfiguración dinámica del sistema.

- Arquitecturas de computadores. La mayoría de las arquitecturas de computadores disponibles no satisfacen los requisitos deseables para su aplicación en sistemas de tiempo real. Las arquitecturas para tiempo real deben proporcionar mecanismos hardware de apoyo al tratamiento de fallas, a los algoritmos de planificación, a los sistemas operativos y a los lenguajes de programación de tiempo real. La topología de interconexión de los procesadores con los dispositivos de entrada/salida y comunicaciones rápidas y seguras, también son factores pendientes de soluciones adecuadas.

- Comunicación. Las características requeridas son predecibilidad, confiabilidad y altas prestaciones.

4.4.1 Clasificación El plazo de entrega es definido como el intervalo de tiempo, en el cual las acciones de los sistemas de tiempo real deben ser ejecutadas. En base a esta definición, los sistemas en tiempo real se pueden clasificar en *Críticos* y *No Críticos*.

- Sistemas de tiempo real críticos En un sistema de tiempo real crítico, las acciones deben ser obligatoriamente realizadas dentro de los plazos de entrega establecidos. La violación de dichos plazos ocasiona graves consecuencias en procesos delicados.

Ejemplos de estos sistemas son los sistemas de control de aviones, satélites, control de equipos quirúrgicos, etc.

- Sistemas de tiempo real no críticos Un sistema de tiempo real no crítico, soporta el incumplimiento en los plazos de las tareas, ocasionando simplemente un funcionamiento un poco degradado del sistema, pero que no trae consigo mayores riesgos.

Como ejemplos se pueden mencionar algunos sistemas multimedia, procesos de producción, adquisición de datos, etc.

- Otros tipos de sistemas Entre los sistemas de tiempo real, se mencionan otros tipos de sistemas, como lo son los Increméntales y los No Esénciales.

    - Increméntales: La respuesta depende del tiempo de computo. Ejemplo de esto sistemas son los que utilizan cálculos iterativos.

    - No Esénciales: Son los sistemas que no presentan restricciones de tiempo real.

4.4.2 Aspectos importantes Los sistemas de tiempo real, pueden estar implementados en software y/o en hardware cumpliendo la misma finalidad, pero generalmente cambiando su manera análisis.

Para la planificación de sistemas en tiempo real, se utiliza mucho el término tarea, que está definida como una actividad interna que no requiere de mensajes externos para iniciar su actividad. La actividad es realizada por un hilo de ejecución interno a ella. Las tareas comúnmente más utilizadas son las Periódicas y las Aperiódicas o esporádicas.

Las características esenciales que se deben tener en cuenta para la planificación de tareas son: Período, plazos de entrega, fase inicial, tiempo de computo, retardo de inicio y tiempo de finalización.

- Período (P): Ejecución regular de las operaciones en intervalos de tiempo establecidos.

- Plazo de entrega (D): Representa el instante de tiempo en que las operaciones del sistema deben de haber finalizado.

- Fase Inicial (Δi): Denota el desfase que tiene el inicio de la primera activación de la tarea del sistema, con respecto al tiempo de inicio.

- Tiempo de computo (C): Es el tiempo que requiere una operación del sistema para su completa ejecución.

- Retardo de inicio (r): Tiempo requerido para que la acción de una tarea inicie su ejecución.

- Tiempo de finalización ([f,f+]): Tiempo en el cual una tarea concluye la ejecución de sus actividades.

5. COMPONENTES BÁSICOS EN EL MODELAMIENTO DE SISTEMAS CONTINUOS DE CONTROL EN AMBIENTES DISTRIBUÍDOS

El modelamiento formal de sistemas se considera un lujo para las grandes compañías, ya que estas pueden invertir mucho dinero realizando investigaciones en el campo con el fin de optimizar sus procesos[16].

Partiendo de que el diseño arquitectural, es una de las más importantes etapas en el desarrollo de un proyecto, se pueden plantear diferentes tipos de estrategias que permitan de una manera simplificada acercar esta fase de diseño formal a la mediana y pequeña industria, con el fin de que éstas se puedan beneficiar de su utilización, en cuanto a la reducción de costos (especialmente por falencias en los diseños implantados) y tiempo, o en la optimización del control procesos.

En este trabajo se han venido dando una serie de conceptos relacionados con el diseño formal, para poder tener un marco conceptual claro sobre lo que se pretende plantear en adelante. La tesis se centra en proponer una estrategia que simplifique el desarrollo de la implementación del nivel arquitectural en sistemas continuos que operen sobre plataformas distribuidos. Tal planteamiento consiste en la presentación de una colección, reducida y práctica, de componentes que por su simpleza permitan modelar fácilmente los sistemas en mención.

5.1 PLANTEAMIENTO DE LA PROPUESTA

El campo abierto para el análisis de los NCS es bastante amplio, razón por la cual se deben establecer límites con el fin de realizar un estudio mejor orientado y más claro. Para esta tesis, los sistemas tenidos en cuenta son los sistemas continuos de control en ambientes distribuidos, dando un énfasis a su aplicación en tiempo real.

Esta propuesta pretende entregar una forma de modelar los sistemas mencionados, para que los proyectos puedan ser diseñados eficientemente. Por medio de la elaboración de modelos formales, se pueden realizar validaciones, para tener una idea del comportamiento real del sistema.

Para cumplir con este objetivo, se realizará la descripción especial de algunos componentes útiles para el modelamiento de NCS con las características descritas. Estos elementos serán representados en HRT-HOOD y en redes de Petri Coloreadas, entregándoles con esto propiedades arquitecturales para el diseño formal.

[16] MARTINEZ, Op.cit.

Para aplicaciones que no tengan restricciones temporales (sistemas no esénciales), los objetos podrían ser representados utilizando UML en lugar de HRT-HOOD. La selección de HRT-HOOD se realizó teniendo en cuenta que este lenguaje posee elementos definidos para aplicaciones que incluyen tiempo real; se estudiarán estas aplicaciones debido a que son las más que más atención demandan al momento de ser modeladas. Otra razón para seleccionar HRT-HOOD es su estrecha relación con lenguajes de programación a nivel Software y Hardware, como lo son ADA y VHDL, respectivamente.

La elección de las redes de Petri Coloreadas se basa en que sus características permiten obtener muchas ventajas en la parte de diseño y validación.

Con la colección de los objetos disponible, un sistema podrá ser modelado utilizando una estrategia que se consta de dos pasos:

- *Modelamiento del sistema en HRT-HOOD:* Consiste en realizar una combinación de los diferentes objetos buscando que el modelo describa correctamente al sistema. En este paso, se utilizará la descripción de los diferentes objetos en HRT-HOOD, teniendo posibilidades de realizar estructuras jerárquicamente para la descripción del sistema.

- *Modelamiento del sistema en Redes de Petri:* Cuando el sistema ha sido modelado en HRT-HOOD, se implementará en Redes de Petri Coloreadas para su posterior validación. Es de observar que estas redes permiten modelar el sistema en forma jerárquica, de tal manera que la representación del sistema visualmente será muy similar a la del modelo realizado en HRT-HOOD.

La recomendación de seguir los pasos en ese orden no es estricta, sin embargo en la práctica resulta muy útil.

En el capítulo 7 se presenta un ejemplo donde se utiliza esta estrategia para el modelamiento de un sistema controlador de nivel.

5.2 CONDICIONAMIENTOS Y RESTRICCIONES

Algunos aspectos que deben ser cuidadosamente tratados para la aplicación de la estrategia se describen a continuación:

5.2.1 HRT-HOOD La utilización de HRT-HOOD por sí solo no representa el análisis formal del diseño. HRT-HOOD representa las características estructurales y funcionales del sistema, pero su *visión* es *estática*. Para completar el análisis formal, se deben utilizar las CPN, por medio de las que el sistema será validado, tanto funcional como temporalmente.

5.2.2 Sistemas a modelar con los componentes Los objetos que se plantearán son especiales para modelar *sistemas de control* continuos en los que el lazo de retroalimentación es implementado sobre una red distribuida. La aplicación en otro tipo de sistemas no ha sido tenido en cuenta en el desarrollo de esta tesis. Además dichos objetos tienen especialidad en representación de sistemas de tiempo real, pero no se descarta su aplicación sobre sistemas que no tengan estas características temporales.

5.2.3 Nivel terminal Las *estructuras jerárquicas* en las *redes de Petri Coloreadas* se manejarán solo hasta el nivel terminal, en el cual, se localizan elementos que funcionalmente no necesitan ser descritos para el análisis de los NCS. En el nivel terminal se encuentran aquellos elementos hardware o software que ya son implementables.

5.2.4 Hilo de control Un modelamiento extremadamente formal, exige que el *hilo de control* sea manipulado separadamente de la información de la tarea, en un llamado de funciones por ejemplo. Para algunos casos, en el modelamiento implementado sobre las redes de Petri Coloreadas es posible que el hilo de control sea devuelto junto con información necesaria para la ejecución de otros pasos.

5.2.5 Planificación La descripción de los aspectos relacionados con la *planificación* de las tareas, estará basada en prioridades estáticas. La razón de esto es que dichos planificadores tienen un mejor comportamiento que los de prioridades dinámicas cuando existen sobrecargas, por esta razón son utilizados en sistemas críticos de tiempo real. Además el planificador de tareas y de red que se estudiaron en este trabajo manejan ese tipo de asignación de prioridades.

5.3 PLANTEAMIENTO DE COMPONENTES

Un sistema de control en red distribuida, puede tener gran cantidad de componentes como por ejemplo, sensores, actuadores, reguladores, PLCs, microcontroladores, PCs, elementos de red, interfaces para acceder a la red, aplicaciones software para control y supervisión, etc. En esta tesis solo se considerarán aquellos componentes con dinámicas discretas que serán implementados en hardware y software, así como también los protocolos de comunicación con las interfaces, más no las interfaces en sí mismas.

Desde este punto de vista, los conversores análogo- digital o digital-análogo, por ejemplo, no han sido modelados debido a que son elementos hardware que no son diseñados en el proceso a modelar, simplemente responden a una orden “convertir”; lo que si necesita ser incluido en el modelo es el programa que manipula las ordenes de conversión o la información recibida, o entregada, desde o hacia estos dispositivos.

Con base en la anterior justificación, se proponen un grupo reducido de componentes para el modelamiento de sistemas continuos de control en redes distribuidas, clasificados en objetos, aplicaciones y elementos generales:

5.3.1 Objetos Estos componentes representan los elementos básicos presentados para los modelos orientados a objetos. En sistemas de tiempo real, y bajo ciertas condiciones, los objetos activos pueden tener comportamientos periódicos o aperiódicos[17]; razón por la cual, serán los únicos objetos presentados en este subgrupo. Son utilizados para modelar las actividades del sistema.

-Objetos Periódicos
-Objetos Periódicos Expropiables
-Objetos Aperiódicos
-Objetos Aperiódicos Expropiables

5.3.2 Elementos generales Pueden ser utilizados en diferentes partes del diseño, para complementar la implementación de las características del sistema.

-Funciones
-Procedimientos
-Selección Múltiple de Datos
-Componente Protegidos Básicos
-Base de Datos
-Barrera

5.3.3 Aplicaciones Son elementos que se han modelado para aplicaciones especificas, en los sistemas de control en tiempo real.

-Red CAN
-Planificador de Tareas

[17] Gonzáles, Op.cit.

5.4 DESCRIPCIÓN DE LOS COMPONENTES

Una breve descripción de los elementos se presenta a continuación, definiendo por sobre todo su campo de acción.

5.4.1 Objetos periódicos Son utilizados para modelar actividades o tareas cíclicas en intervalos de tiempo constantes. Son objetos que requieren ser inicializados para definir parámetros de periodicidad y de funcionamiento interno de sus actividades. En aplicaciones de tiempo real, por lo general, en su inicialización incluyen una etapa de sincronización con otros sistemas. En condiciones de funcionamiento normal, sus actividades no pueden ser detenidas. En los NCS son especiales para sistemas de muestreo, supervisión periódica de estados, u otros sistemas que permitan la utilización de sus características.

5.4.2 Objetos periódicos expropiables Reúnen todas las características funcionales y temporales de los objetos periódicos comunes, pero a diferencia de estas ultimas, la ejecución de las actividades de estos objetos puede ser interrumpida. Al igual que el anterior objeto necesitan ser sincronizado inicialmente, para aplicaciones de tiempo real.

Aunque es un objeto periódico más general que el anterior, su aplicación es especializada en los sistemas monoprocesador planificables en tiempo real. En estos casos, el planificador elige cual tarea tiene mayor prioridad para continuar su ejecución, si una tarea (del tipo de este objeto) debe suspender sus actividades, le será notificado y ésta será expulsada del procesador, reteniendo el punto de su hilo de control donde fue suspendida, para continuar su ejecución cuando se le autorice nuevamente, sin necesidad de reiniciar todas las actividades

Como se mencionó en el capitulo 4, existen características que hacen que ciertas tareas no puedan ser planificables por métodos matemáticos convencionales, y deban ser especificadas por otros métodos como las redes de Petri. Para el caso de las tareas periódicas y aperiódicas de cualquier clase, hay una característica esencial que se puede convertir en un inconveniente para los métodos matemáticos de planificación: Posibilidad de invocar otros bloque funcionales, como por ejemplo bases de datos, funciones, procedimientos e inclusive a tareas aperiódicas. Es un problema para los métodos de planificación mencionados, en vista de que ellos suponen independencia entre tareas.

-Para sistemas planificados, a los objetos periódicos expropiables se les puede adicionar características de objeto protegido, permitiendo que el procesador interprete sus acciones de tal manera que evite aplicar expropiación. Los objetos protegidos pueden afectar la planificación del sistema en cuanto al manejo las prioridades, ocasionando posiblemente un efecto de inversión sobre estas ultimas. La solución de este tipo de conflictos no está soportada por los métodos matemáticos de planificación.

5.4.3 Objetos aperiódicos Los objetos aperiódicos o esporádicos, inician la ejecución de sus actividades cuando reciben una orden. Aunque tienen un hilo de ejecución, no tienen actividad propia. Estos objetos se diferencian de una función o de un procedimiento, en que al igual que los periódicos, pueden realizar llamados a otros elementos.

5.4.4 Objetos aperiódicos expropiables Son objetos aperiódicos que pueden ser interrumpidos en su ejecución. Son aplicables a sistemas monoprocesador de tiempo real, ya que pueden ser utilizados con efectos de planificación. Estos objetos pueden tener una fase de inicialización para la configuración del sistema o la sincronización del mismo; este proceso se convierte en un requisito obligatorio (si el sistema no está inicializado, no ejecuta sus acciones).

5.4.5 Funciones Elementos que al ser invocados realizan determinadas acciones y retornan un valor, y de paso el hilo de control al invocador. Pueden recibir parámetros del objeto que realiza el llamado. Generalmente son inicializadas. Se diferencian de los objetos aperiódicos en que no tienen plazos de entrega ni manejo del hilo de control del proceso, y tampoco pueden realizar invocaciones a otros objetos que no sean funciones o procedimientos. Son una representación de los objetos pasivos.

5.4.6 Procedimientos La única diferencia con las funciones es que no retornan ningún valor, retornan solo el hilo de control al objeto invocador.

5.4.7 Selección múltiple de datos Permite a alguna tarea, función o procedimiento, escoger entre una o más opciones, la requerida para realizar algún paso de su proceso. Este objeto se implementó con un uso especifico, seleccionar un dato, bien sea numérico o de otro tipo de variable. Como condición, se establece que la opción seleccionada se encuentre disponible cuando se selecciona.

5.4.8 Componente protegido básico Se encarga de proteger información. El acceso a este elemento, generalmente se realiza por medio de funciones o procedimientos. Si se accesa por medio de funciones, el objeto es solo para la lectura de datos, la cual se puede realizar de forma concurrente. Cuando el acceso es por medio de procedimientos, además de leer los datos protegidos, también se pueden cambiar o escribir; no se puede acceder de manera concurrente.

5.4.9 Base de Datos Una aplicación especifica de los componentes protegidos es la base de datos, elemento que se puede implementar reduciendo el modelo del componente protegido a accesos solo con procedimientos. En un instante de tiempo, la base de datos puede ser escrita ó leída, con lo cual se asegura que el dato no sea modificado mientras se está leyendo, ya que esto causaría como mínimo una inconsistencia en el valor (si se puede leer). El acceso a la base de datos no se puede realizar de manera simultánea.

5.4.10 Barrera Es un tipo de objeto protegido que se utiliza para sincronizar tareas, bien sea inicialmente o dentro de la ejecución, por medio de la retención de los hilos de control de las tareas hasta que se cumple la condición de activación. Esta condición es la llegada de la cantidad de tareas esperadas.

5.4.11 Red CAN Para soportar las aplicaciones industriales en sistemas distribuidos, aparecen a principios de la década de los años noventa los buses de campo, que vienen a definir protocolos específicos para la comunicación en redes industriales.

Para el desarrollo de aplicaciones de automática en la industria se han utilizado varios protocolos y buses de campo entre los que se pueden mencionar Ethernet, Token Ring, Device Net, Profibus, etc. Para esta tesis, en donde se desea analizar aplicaciones con requerimientos hard (fuertes) de tiempo real, se ha seleccionado como protocolo de comunicación a CAN, el cual tiene un comportamiento predecible y permite ingresar fácilmente nuevos nodos al sistema, además su implementación es económica, razón por la cual su aceptación en el medio industrial ha sido grande.

El componente modela varias de las características del protocolo CAN, entre las que se encuentran, la inclusión de los retardos de la red, retardo del mensaje en ser transmitido y manejo de prioridades. El sistema es soportado por medio de tramas que incluyen parte de la composición real del protocolo. Este elemento puede ser clasificado como objeto protegido, ya que no permite que el dato presente en la red pueda ser expulsado de ella hasta que su transmisión no sea terminada.

5.4.12 Planificador de Tareas Para la implementación a nivel arquitectural de sistemas computacionales en tiempo real, es necesario realizar un proceso de asignación de prioridades de las tareas y con base en esto y otros factores, como los tiempos de computo, finalización, etc., se realiza un análisis para determinar si el sistema funcionará correctamente dentro de los plazos establecidos, este proceso se conoce como planificabilidad.

El planificador es el sistema encargado de controlar el acceso al procesador. Aunque es un elemento que funcionalmente está implícito en los lenguajes y sistemas operativos de tiempo real, es necesario definir por medio de un modelo ciertas características operacionales del mismo, para lograr verificar el funcionamiento de las tareas en tiempo real. En esta tesis, se presenta el modelo de un planificador basado en prioridades estáticas, por medio del cual se pueden realizar validaciones en los análisis de planificabilidad para sistemas que no sean fácilmente planificables por otros métodos (matemáticos como rate monotonic), y que cumplan con la característica de las prioridades que éste maneja.

6. REPRESENTACIÓN DE COMPONENTES A NIVEL ARQUITECTURAL

En el capitulo anterior se plantearon algunos componentes fundamentales para realizar el modelamiento de sistemas continuos de control industrial en ambientes distribuidos, y se presentó una estrategia de diseño para la representación a nivel arquitectural de dichos sistemas por medio de los componentes expuestos.

Como primer paso se adoptó el modelamiento del sistema utilizando HRT-HOOD, con el propósito de tener una visión adecuada del proceso que se pretende tratar. Posteriormente se utilizan las Redes de Petri Coloreadas para realizar las validaciones temporales y funcionales del sistema.

Los componentes representados en HRT-HOOD, de manera individual, se presentarán junto con sus modelos en CPNs en este capitulo. Estas descripciones son bastante simples, pero muy útiles para tener fácil referencia de dichos elementos. La aplicación de HRT-HOOD se amplía cuando se comienzan a mezclar los diferentes objetos, adicionando los flujos e información, de excepción, etc.

La descripción de algunos componentes en redes de Petri está soportada en trabajos como el presentado en ([18]), pero en algunos casos se realizan modificaciones para aplicarlos a los sistemas hacia los que está encaminado este proyecto.

El diseño de los diferentes elementos se realizó sobre un software especializado en el modelamiento de redes de Petri Coloreadas: *Design/*CPN. Esta herramienta fue seleccionada debido a que ofrece un entorno de modelamiento muy apropiado para realizar el diseño de los sistemas. Cuenta con una semántica definida, tanto en la implementación de todas las características de las redes de Petri en sus principios fundamentales, como para la utilización de código, ya que está basado en un estándar conocido como SML´97, que reúne las condiciones necesarias para que una aplicación tenga los suficientes fundamentos en su composición. Design/CPN, es fácil de manipular, y reúne muchas característica aplicativas a diferentes tipos de análisis.

Los componentes modelados en las redes de Petri se introducirán en dos niveles jerárquicos. El primero, representa la visión externa del elemento, mostrando solamente una transición, y unos lugares de salida y de entrada, para representar los servicios que ofrece o demanda. La representación del elemento en HRT-HOOD es fácilmente relacionada por el modelador con este primer nivel jerárquico del componente. Como segundo nivel en la jerarquía, se describe la forma en que el componente está

[18] Gonzáles, Op..cit.

implementado. Este nivel permite analizar y manipular las características del objeto para adecuarlo a la aplicación especifica o sistema a modelar.

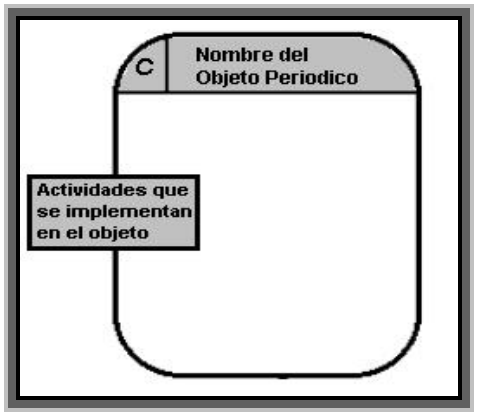
En el Apéndice B se presentan algunos aspectos sobre la interpretación de las redes de Petri Coloreadas implementadas sobre Design/CPN. Se recomienda leer ese apartado para entender bien la estructura de las CP-nets planteadas en lo que sigue.

6.1 OBJETO PERIÓDICO

Uno de los componentes más utilizados en el modelamiento de diferentes clases de sistemas es el objeto periódico. Modela actividades cíclicas con un periodo constante de ocurrencia. A diferencia de otros objetos, estos pueden llevar hilos o secuencias de control, por lo cual son aplicables para la representación de tareas. En su estructura funcional, se puede mencionar que para el desarrollo de sus actividades pueden invocar operaciones ubicadas en otros objetos, al igual que pueden hacer llamados a funciones y/o procedimientos. En su proceso de inicialización se definen parámetros operacionales como declaraciones de variables, inicializaciones de las salidas del sistema, y otros factores como las sincronizaciones, las cuales son un requisito indispensable para el modelamiento en tiempo real. Sin embargo la sincronización no es un requerimiento obligatorio para todos los componentes del sistema.

En la figura 23, se presenta el modelo del objeto periódico en HRT-HOOD.

Figura 23. Objeto Periódico en HRT-HOOD

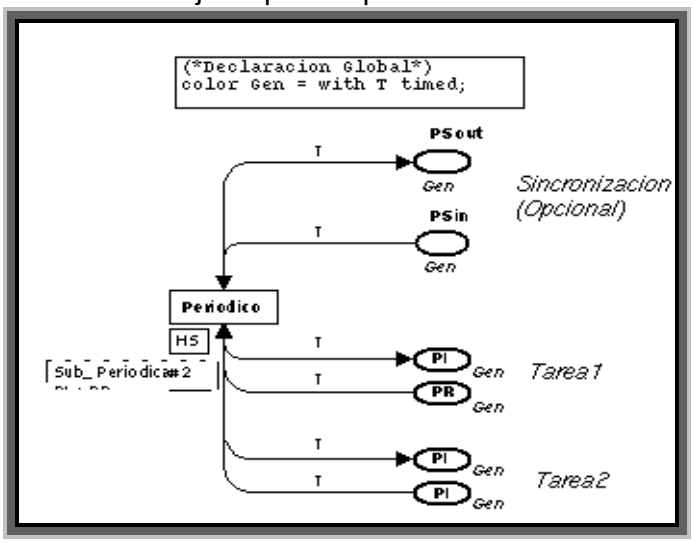


En la interfaz se describen algunas operaciones del objeto, como por ejemplo, lectura de sensores, actualización de información, etc.

La representación en redes de Petri Coloreadas del primer nivel está presentada en la figura 24. La transición *periodico* representa de manera general al objeto periódico. Los

lugares *PSout* (Place Sincronización) y *PSin* son utilizados para indicar que el sistema está listo para ser sincronizado y está a la espera de una orden para continuar su ejecución. Esta sincronización, como se indica en la figura es opcional. También se implementa una interfaz para llamar tareas, a través de los lugares *TI* (Transition Invocación) y *TR* (Transition Retorno). Esta interfaz sirve de igual manera para realizar llamados a funciones o procedimientos. Para la presentación del objeto la interfaz contiene el llamado a dos tareas, sin embargo el sistema puede ser modelado para la cantidad de tareas, funciones o procedimientos que el sistema amerite, e inclusive para la ausencia de estos casos.

Figura 24. Objeto Periódico: nivel jerárquico superior

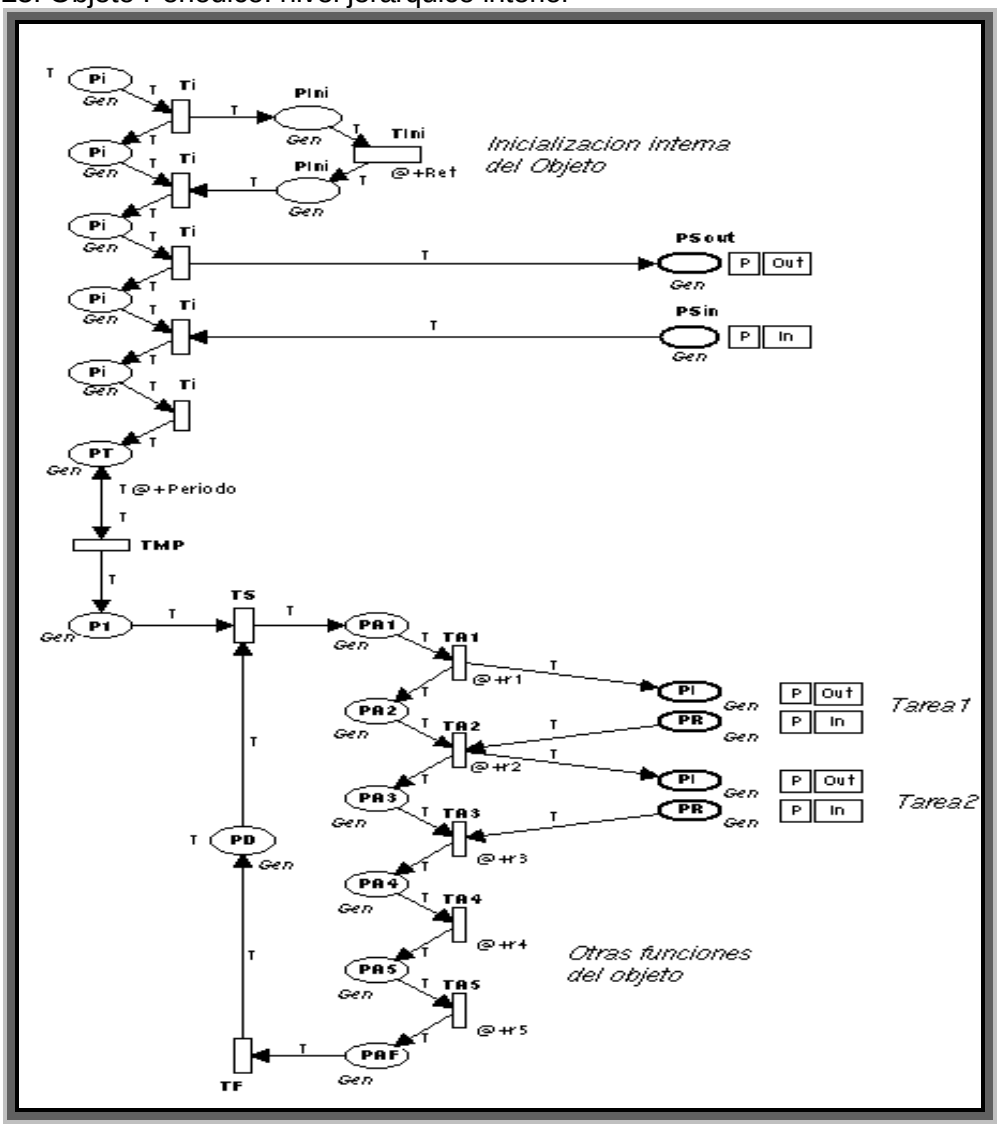


La figura 25 muestra la implementación del componente en su segundo nivel. Se puede observar que el objeto contiene una secuencia de lugares (*Pi*, Place de Inicialización) y eventos (*Ti*, Transitions de Inicialización) que representan la inicialización del sistema. A través de este proceso de inicialización, el objeto puede establecer parámetros inherentes a la operación de su tarea; algunos de esos pueden ser los valores del periodo de ejecución, inicialización de variables del sistema, etc. En el modelo, el período de muestreo corresponde a un valor numérico que debe ser reemplazado en el arco que relaciona a la transición *TMP* (Temporización) y el lugar *PT* (Place Temporización), más específicamente donde se indica como *Periodo*.

Siguiendo la secuencia de inicio planteada a manera de ejemplo en el modelo, se encuentra después de la ejecución de la primera transición *Ti* del sistema, un lazo compuesto por dos lugares *Pini* (Place Inicializador) y la transición *Tini* (Transition Inicializadora), representando la inicialización de los parámetros mencionados. Este paso trae consigo el respectivo retardo asociado a la fase de inicialización. Luego de que el sistema ha sido inicializado, puede incluirse si es necesario una etapa de sincronización. para este fin el sistema tiene dos lugares que permiten implementar dicho proceso, *PSout* y *PSin*. PSout le indica al elemento de sincronización que su fase de inicialización ha sido

terminada, y que está listo para recibir la orden de inicio. Esta orden es recibida a través del lugar *PSin* y le permite a la tarea iniciar su actividad.

Figura 25. Objeto Periódico: nivel jerárquico interior



Antes de que el objeto realice su primer ciclo, se pueden realizar algunas acciones que el modelador estime convenientes. La actividad periódica es iniciada cuando llega una marca al lugar *PT*, activando la transición *TMP*, la cual envía un token a *P1* y otro token a *PT* retardado en el valor indicado por *Periodo* (previamente reemplazado por el valor de retardo deseado).

Cuando se encuentra una marca presente en el lugar *P1*, se da inicio a la ejecución de las actividades implementadas en el objeto. *TS* (Transition Start) observa si el objeto puede

atender la orden de ejecución periódica, activándose solamente si el lugar que indica dicho estado de disponibilidad, *PD* (Place objeto Disponible), tiene la marca correspondiente, de lo contrario, la orden no será atendida hasta tanto el objeto se desocupe. Por esta razón, cuando se están utilizando este tipo de objeto, es primordialmente necesario saber cuanto se tardará la ejecución de todas las actividades del objeto, antes de definir el valor del periodo.

Si el objeto se encuentra disponible, y hay una orden de ejecución de actividades, la transición *TS* se activará, dando paso a la operación de las mismas, a través de la ejecución de las diferentes transiciones comprendidas entre los lugares *PA1* (Place Actividad 1) hasta *PAF* (Place Actividades Finalizadas); en el modelo se observan cinco actividades, pero en las implementaciones pueden ser menos o más actividades dependiendo del proceso. En la figura (figura 25) se han modelado algunas operaciones dentro del objeto, que son un ejemplo de las actividades que se pueden incluir en ellos. Con una marca en *PA1* se activa la transición *TA1* que en su ejecución, genera una marca en *PI*, para hacer el llamado de alguna operación que se encuentra externa a él (Invocación de tarea, procedimiento o función), y coloca otra marca en el lugar *PA2*, para que se interprete el sitio en que el modelo está siendo ejecutado. Cuando la operación invocada se finaliza, el hilo de control retorna al objeto, y se le entrega al objeto un resultado de la operación, o la confirmación de que se ejecutó correctamente. Después de terminada la invocación a la primera tarea, función o procedimiento, se continúa con una segunda invocación, provocada por la ejecución de la transición TA2. Finalizada esta ultima, se realizan una serie de actividades (desde *TA3* hasta *TA5*) que interpretan operaciones de computo interno del objeto. Los retardos asignados a las transiciones *TA1* a *TA5* representan el tiempo invertido para la realización de acciones de invocación de tareas externas, o el cumplimiento de las funciones internas. Los valores de estos retardos deben ser previamente establecidos y reemplazados en el modelo.

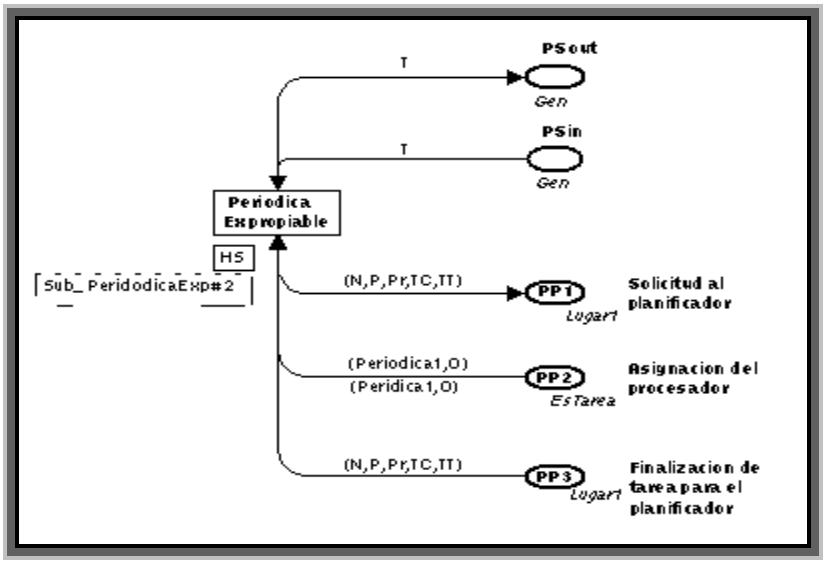
Cuando todas las actividades son ejecutadas, el objeto protegido estará disponible para atender otras ordenes de ejecución; después de la ejecución de *TF* (Transition Finalización de actividades), el lugar *PD*, será nuevamente marcado para indicar lo anterior.

Finalmente es necesario aclarar que en el modelo se han utilizado solo marcas generales, como se puede observar en la declaración de la figura 23, pero en los procesos a modelar se pueden utilizar distintos tipos de colores para estos tokens. Por ejemplo, si después de la invocación de una operación, es devuelto un dato, el color utilizado para ese token corresponde al valor recibido (pueden ser enteros, caracteres, etc).

6.2 OBJETOS PERIÓDICOS EXPROPIABLES

Estos objetos cumplen con todas las características mencionadas para los objetos periódicos comunes. Se diferencian en que su ejecución puede ser detenida o expropiada. Su modelo para el primer nivel en CP-nets se presenta en la figura 26.

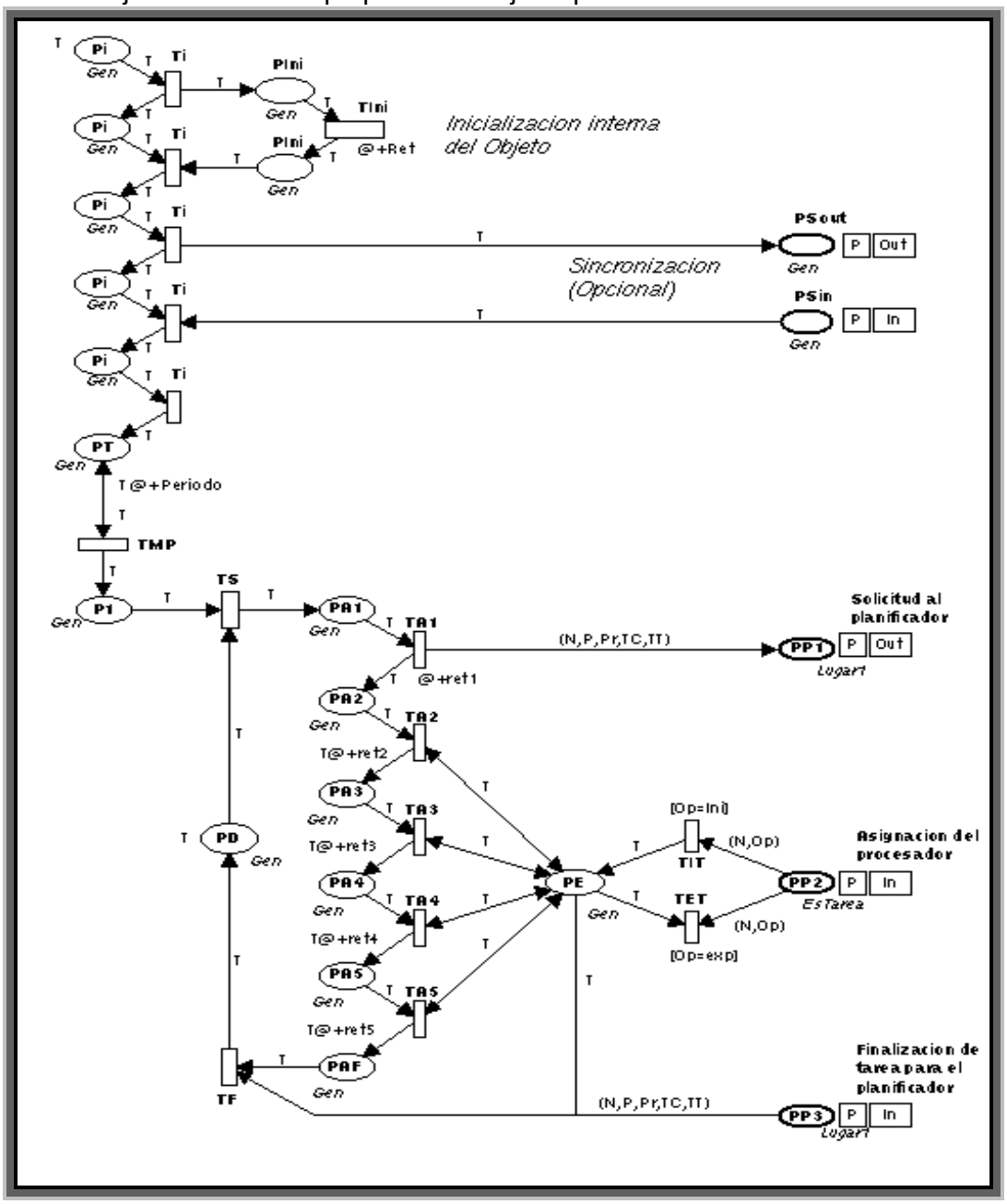
Figura 26. Objeto Periódico Expropiable: nivel jerárquico superior



En su inicialización a este objeto se le configuran, además de todas las variables normales del objeto periódico, las características de una tarea, como por ejemplo, los plazos y las prioridades.

Al modelo desarrollado para el objeto periódico se le ha adicionado un campo para realizar las operaciones de la relación Objeto-Planificador. Por medio del lugar *PP1* (Place Planificador 1) se realiza la solicitud del procesador al planificador. Cuando el planificador de tareas decide que la tarea debe ser ejecutada, se coloca una marca en *PP2*, en la cual debe ir especificado que la tarea puede ser iniciada. Cuando la tarea ha cumplido con su ciclo de actividades, en el lugar *PP3* se coloca una marca que representa la expulsión de la tarea del procesador.

Figura 27. Objeto Periódico Expropiable: nivel jerárquico inferior



En la figura 27, se muestra el segundo nivel del objeto periódico expropiable. Su estructura es muy similar la del objeto periódico, exceptuando la implementación de la comunicación con el planificador y la ubicación de los retardos asociados a cada una de las actividades que el objeto realiza. A continuación se tratará el análisis de las modificaciones y adiciones realizadas al modelo de la tarea periódica común.

Para efectuar cada una de las actividades del objeto, es necesario verificar si el procesador está asignado a la tarea. Esto se realiza por medio del lugar *PE* (Place posible Ejecución), el cual, cuando se encuentra marcado indica que se puede ejecutar cualquier actividad que realice la consulta, pero si no hay token en ese lugar, significa que el procesador no está asignado a la tarea, y de está manera este lugar bloquea la ejecución de las actividades. Es de observar que si una tarea es expropiada durante la secuencia de actividades, la actividad que debía ejecutarse en el paso siguiente (según el hilo de control) es detenida, y cuando la tarea pueda seguir su ejecución normal, el hilo de control partirá desde el punto donde fue interrumpido, o desde la actividad que fue detenida.

El inicio y la expulsión de la tarea, son controlados desde el planificador, y comunicados al objeto por medio del lugar *PP2*. Las transiciones *TIT* (Transition Inicio de tarea) y *TET* (Transition Expulsión de Tarea) seleccionan la opción indicada por la marca que llegue a *PP1*, con el fin de reflejar la orden (de inicio ó expulsión) sobre el lugar *PE*. Si el planificador envía una marca al lugar *PP1* que indica que la tarea puede iniciar, o continuar su ejecución, la transición que se habilita es *TIT*, y en su ejecución colocará una marca en *PE*. En el caso contrario (si la orden que llega a *PP1* es de expulsión), la transición *TET* se habilita y con su disparo removerá el token existente en *PE*.

En comparación con el modelo del objeto periodico, los retardos ocasionados por la ejecución de las actividades, se ubican ahora entre las transiciones *TA* (*TA2*, *TA3*, ...) y los lugares *PA* (*PA2*, *PA3*, ...), y no en las transiciones *TA*. Esto se realiza para no afectar con retardos a la marca que se retorna al lugar *PE* , después de la ejecución de cada una de las transiciones mencionadas. La transición TA1 incluye el retardo ocasionado al realizar la solicitud al procesador.

Los formatos de los tokens que reciben los lugares *PP1*. *PP2* y *PP3*, se especifican a continuación:

-*PP1* y *PP3 :* (N,P,Pr,TC,TT), donde N es el nombre o identificador de la tarea, P es la prioridad, Pr indica si la tarea es protegida o no (en el caso de este objeto tendrá un valor de *false*), TC es el tiempo de computo de la tarea, y TT es el tiempo transcurrido en la ejecución de la misma, valor necesario para recuperar el estado de la tarea después de haber sido expropiada o para que el planificador identifique el momento de la expulsión de la tarea por su finalización.

-*PP2*: (N,O), N es el nombre de la tarea y O representa el tipo de orden que se ejecutará (inicio de tarea o expropiación de la misma)

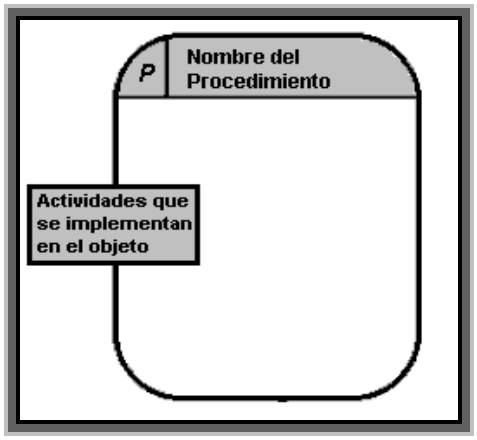
Una descripción más detallada sobre el formato de las marcas y la forma en que estos tokens operan dentro de la planificación del sistema, es presentada en la descripción del modelamiento del planificador.

Aunque en el modelo no se han representado los llamados a operaciones externas, estas pueden ser implementas de igual forma a la ilustrada en el objeto periódico simple.

6.3 OBJETOS APERIÓDICOS

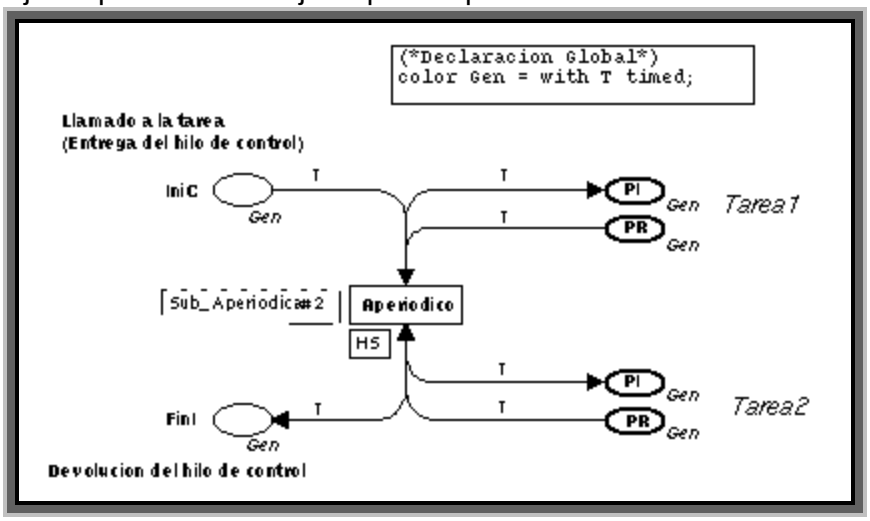
Estos objetos responden a estímulos externos (invocaciones), ejecutando una secuencia de pasos y retornando, en algunas ocasiones, un valor al objeto invocador, a manera de respuesta. Su representación en HRT-HOOD se presenta en la figura 28.

Figura 28. Objeto Aperiódico en HRT-HOOD



La representación en redes de Petri Coloreadas, en su nivel superior se presenta en la figura 29.

Figura 29. Objeto Aperiódico: nivel jerárquico superior

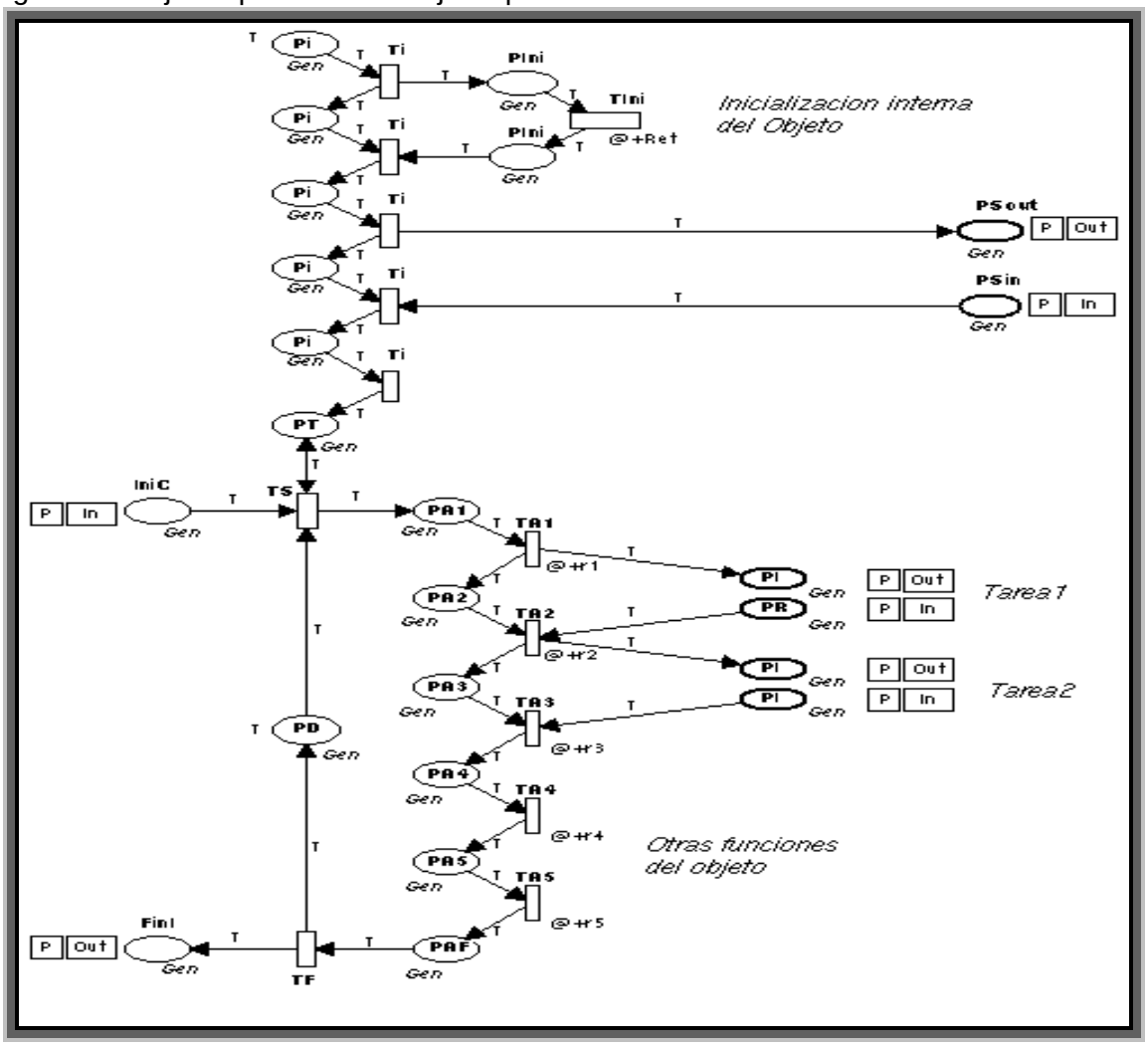


El objeto Aperiódico recibe el hilo de control de un proceso externo por medio del lugar *IniC* (IniCialización) y puede manipularlo como si fuera un proceso completamente independiente, llamando a otros objetos, funciones y operaciones (a través de los lugares *PI* y *PR*), según lo indiquen las actividades a efectuar. Al final de su ejecución, retorna el hilo de control al objeto que realizó el llamado, por medio del lugar *Finl* (Finalización). El hilo de ejecución retornado puede estar acompañado de datos, o puede ser independiente.

Los tokens que se manejan en la red son todos generales (definidos por la declaración en la figura 29), pero en las aplicaciones pueden contener datos, para retornar el hilo de control con información, por ejemplo.

La figura 30 muestra una visión más detallada del objeto, por medio del segundo nivel de jerarquía para el objeto. Como se puede observar la relación estructural entre este objeto y el objeto periódico es grande, sin embargo funcionalmente utilizan los componentes de la red de Petri de manera diferente.

Figura 30. Objeto Aperiódico: nivel jerárquico inferior



La Inicialización y sincronización de este objeto son opcionales, y dependen de cómo se implementan las actividades. Por ejemplo, la Inicialización es requerida para procesos que utilicen valores definidos, ubicar actuadores en determinada posición, inicializar periféricos, etc. Si no es necesaria esta fase de Inicialización y sincronización, simplemente se deben sustraer del modelo los lugares , correspondientes a la secuencia de inicio: Lugares *Pi* , transiciones *Ti*, y todos los elementos asociados a estos dos componentes.

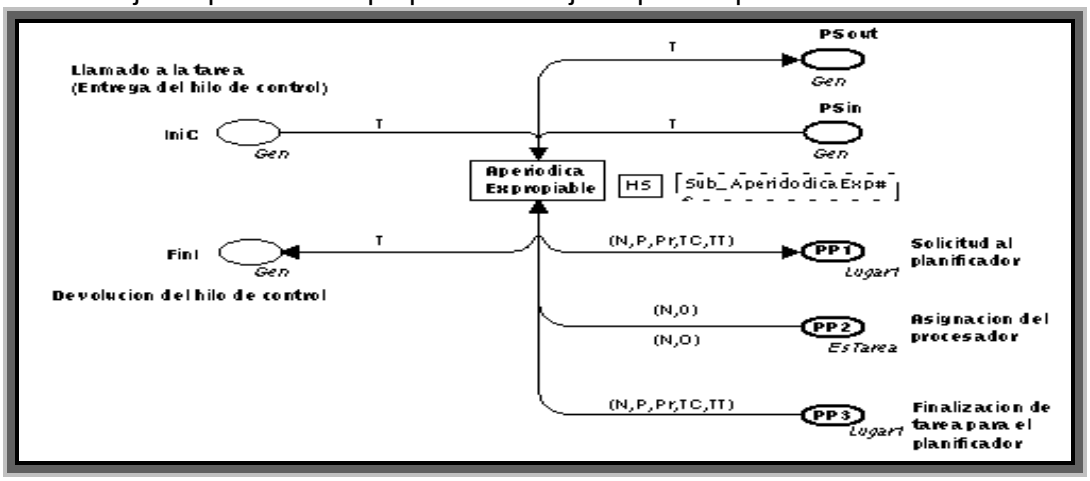
Cuando la Inicialización es incluida, ningún proceso externo podrá iniciar el ciclo de trabajo del objeto Aperiódico, hasta que la Inicialización sea concluida. Esta propiedad se implementa por medio del arco bidireccional que une al lugar *PT* con la transición *TS*. Entonces en este objeto, la transición *TS* tiene la doble función de verificar si el objeto está disponible y si el este ha sido inicializado.

La ejecución de la secuencia de actividades, para realizar operaciones internas y externas del objeto es la misma que la presentada en la figura 25 para los objetos periódicos. Solo se presenta una pequeña diferencia estructural a la finalización, ya que en este objeto, la transición *TF* no solo indica que el objeto está disponible nuevamente (colocando una marca en *PD*), sino que también retorna el hilo de control al objeto que realizó el llamado, por medio del token adicionado a *Finl*.

6.4 OBJETOS APERIÓDICOS EXPROPIABLES

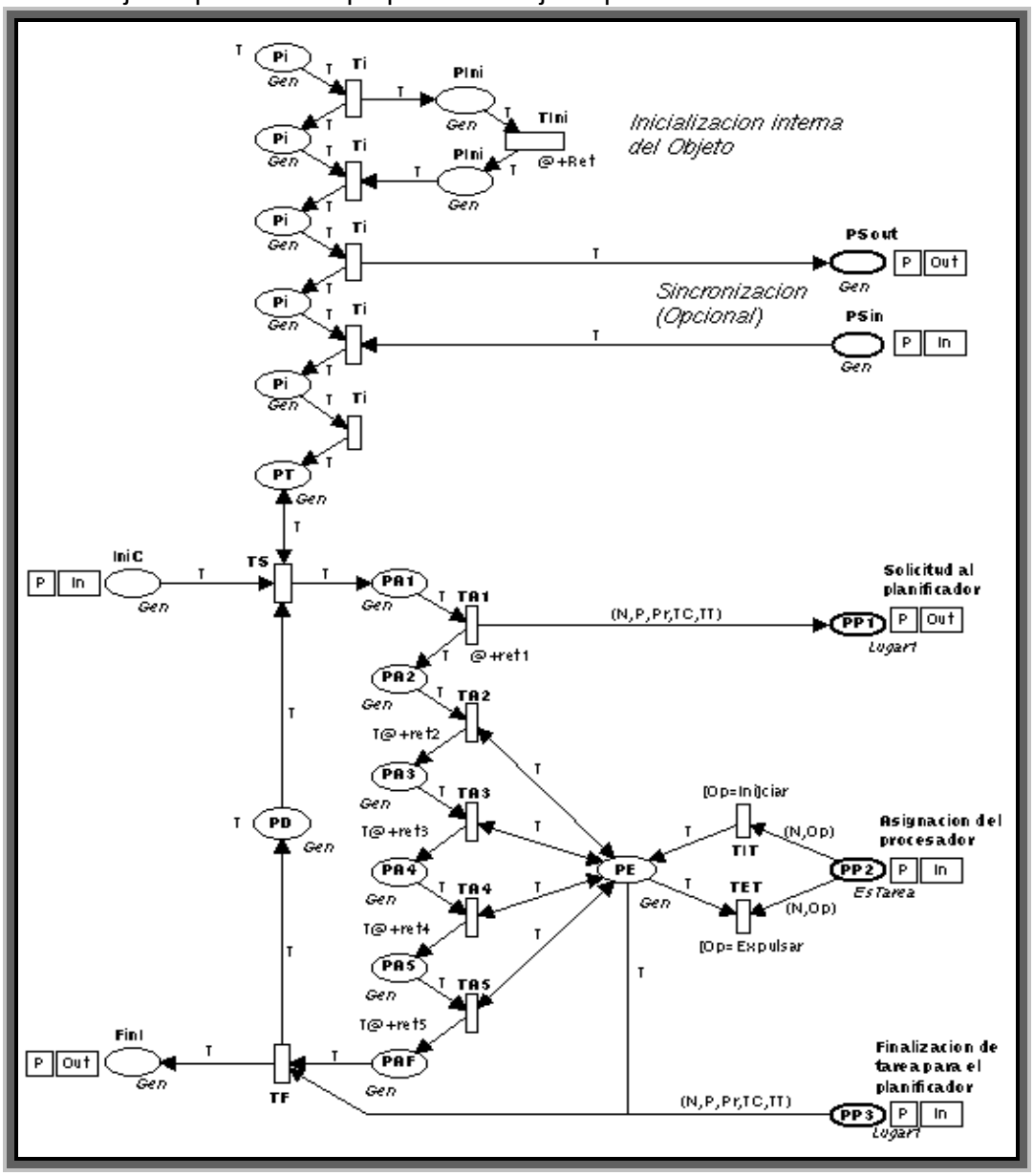
Debido a sus características de expropiación, se puede utilizar para sistemas de tiempo real planificables. En estos casos, la Inicialización es requerida para definir parámetros de tarea como plazos de entrega, prioridades, etc. La sincronización con otros objetos es muy importante, con el fin de determinar los momentos exactos en que el sistema está disponible para empezar su operación.

Figura 31. Objeto Aperiódico Expropiable: nivel jerárquico superior



La figura 31, ilustra el primer nivel de análisis del objeto. La diferencia entre este modelo y el del objeto aperiódico, es la presencia de la interfaz de comunicación con el planificador de tareas. En la figura 32, se presenta el segundo nivel del objeto.

Figura 32. Objeto Aperiódico Expropiable: nivel jerárquico interior.



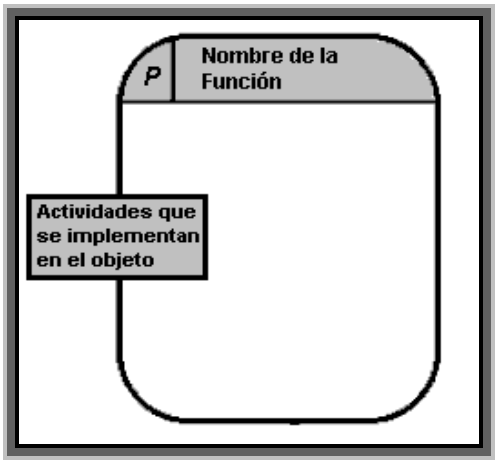
La estructura para el llamado de objetos externos se realiza de la misma forma como se expuso en el objeto Aperiódico normal. Se adiciona el condicionamiento para la ejecución de cada actividad, dependiente del estado del procesador con respecto al objeto. El estudio de esta parte se presentó en el objeto periódico.

6.5 FUNCIONES

Las funciones son similares a los objetos aperiódicos, pero a diferencia de estos últimos, no pueden manejar hilos de control con tanta libertad. Las funciones son secuencias de código u operaciones que son invocadas por tareas, e inclusive pueden recibir llamados de otras funciones o de procedimientos. La anterior característica permite considerarlas como objetos pasivos. Como característica de los objetos pasivos, las funciones no pueden realizar llamados a ningún tipo de objeto, a menos que éste sea pasivo.

La forma en que puede ser modelada una función en HRT-HOOD es la presentada en la figura 33. En la interfaz se definen las actividades que la función puede realizar.

Figura 33. Función en HRT-HOOD



La figura 34, presenta el primer nivel del objeto, en redes de Petri. Se observa que en comparación al objeto aperiódico, la interfaz de invocación a operaciones externas, no incluye tareas, porque, como se mencionó, las funciones solo pueden hacer llamadas a funciones y procedimientos. Las funciones por lo general retornan valores después de su invocación.

Figura 34. Función, Procedimiento: Nivel jerárquico superior

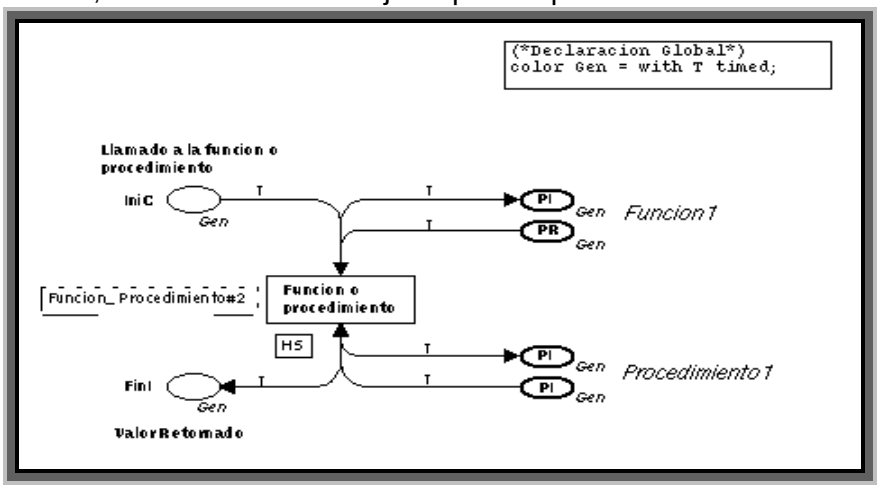
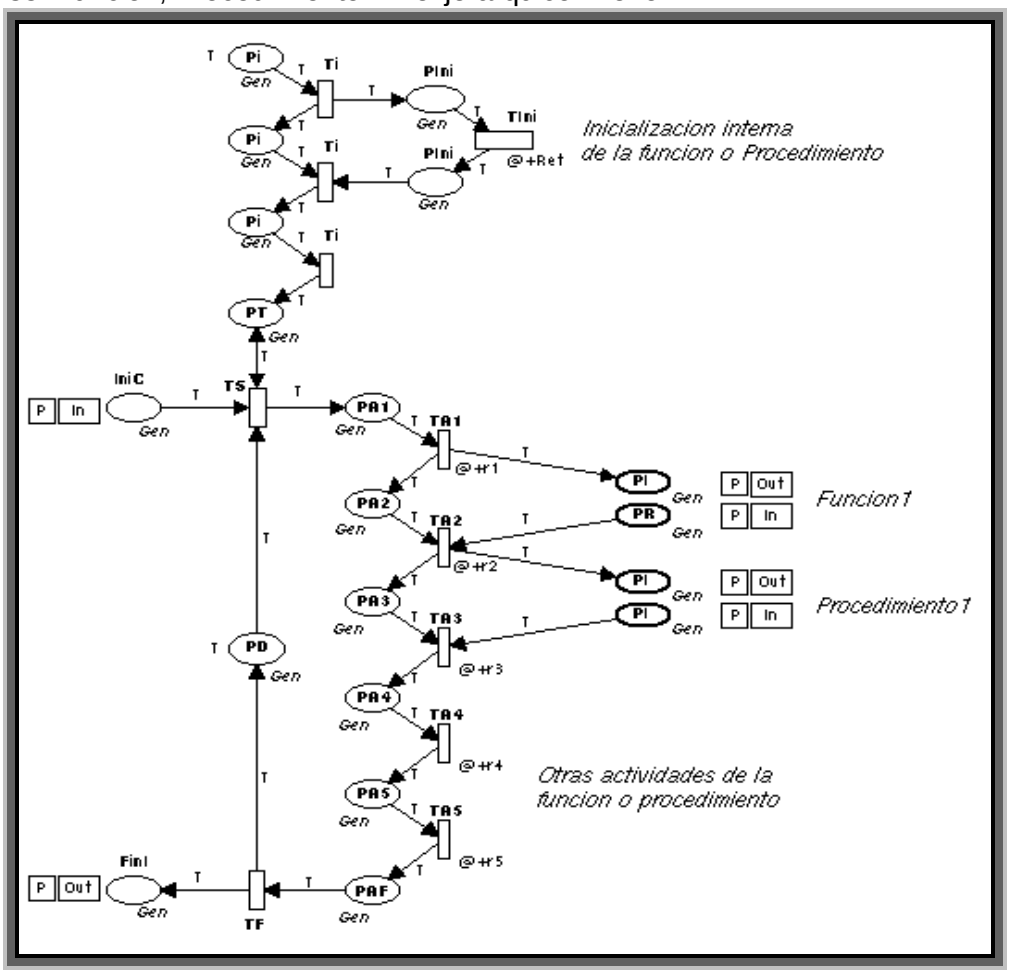


Figura 35. Función, Procedimiento: nivel jerárquico inferior

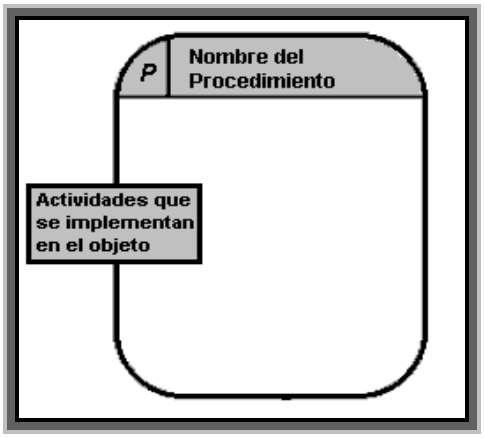


La estructura interna se presenta en la figura 35. La diferencia más notable con respecto a los objetos aperiódicos está en la clase de objetos que puede invocar. Algunas funciones son inicializadas para definir variables. Debido a que no son tareas, la fase de sincronización se puede ignorar.

6.6 PROCEDIMIENTOS

La diferencia con las funciones, es que los procedimientos no retornan valores a final de su ejecución, simplemente devuelven el hilo de control al objeto invocador. En la figura 36 se muestra la representación del objeto en HRT-HOOD.

Figura 36. Procedimiento en HRT-HOOD

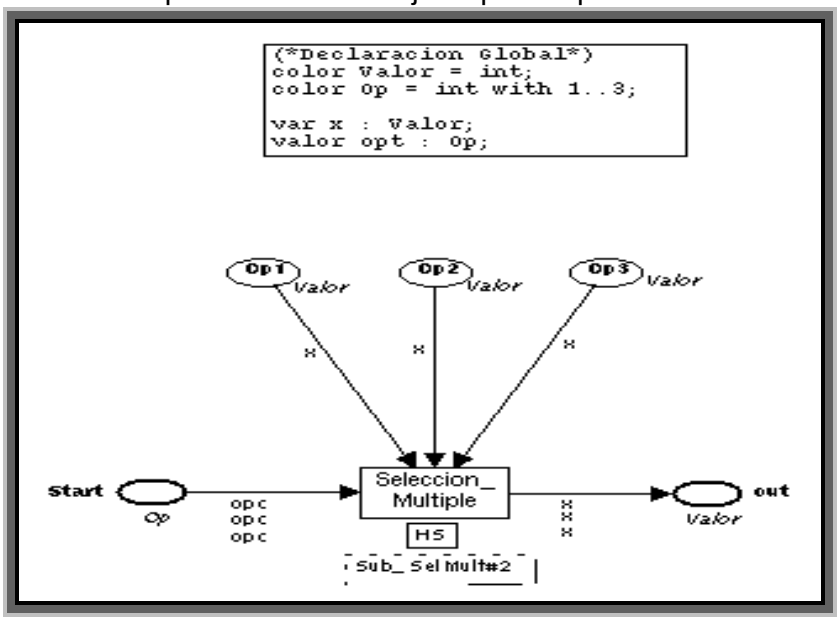


Su modelo en redes de Petri y su funcionamiento son idénticos a los de las funciones. Las ilustraciones de las redes de Petri Coloreadas se ilustran en las figuras 34 y 35.

6.7 SELECCIÓN MÚLTIPLE DE DATOS

Este elemento puede seleccionar un dato entre varias opciones. En la figura 37. se ilustra el primer modelo en CP-nets de este componente. Este modelo está diseñado para seleccionar datos de tipo entero, pero puede modificar en la declaración para adaptarlo a otras necesidades: En la línea en la declaración que define el *color Valor*, se modifica el tipo de color asignado. Por ejemplo se cambia a *int* por *bool*. Para trabajar con valores booleanos en lugar de enteros.

Figura 37. Selección Múltiple de Datos: nivel jerárquico superior



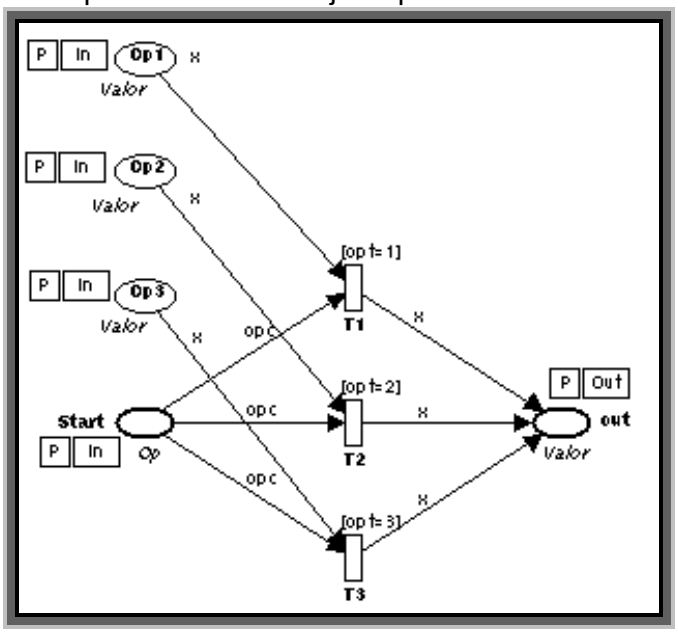
Las diferentes opciones se representan por los lugares de entrada *Op1*, *Op2* y *Op3* (Opciones *Op*) que son solo tres, como ejemplo de este modelo, La cantidad de opciones se puede ampliar o reducir, por medio de la adición o substracción de lugares *Op* y de la modificación (en la declaración) del color *Op* que en este modelo se encuentra configurado para seleccionar entre tres opciones.

En la figura 38, se amplía la visión sobre este componente. Su funcionamiento es muy sencillo.

Cada opción (lugar *Op*) tiene un identificador asociado, desde el momento en que se relaciona con la transición que evaluará su selección (*T1, T2, o T3* para este modelo). Por ejemplo, la opción *Op1* está relacionada con la transición *T1* que indica que está opción será seleccionada si llega una marca con valor de 1 al lugar *Start*.

La descripción funcional es la siguiente: Al lugar *Start* llega el número de la opción que se desea seleccionar, y por medio de los guardas (condiciones para su ejecución) de las transiciones *T1, T2, T3* se activa la transición correspondiente a la opción especificada en *Start*. La acción asociada al disparo de la transición activa, produce como resultado que el valor presente en el lugar de entrada de la transición sea reproducido en la salida del componente *out*.

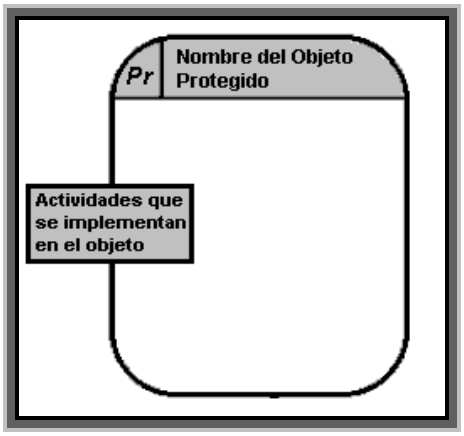
Figura 38. Selección Múltiple de Datos: nivel jerárquico inferior



6.8 COMPONENTE PROTEGIDO BÁSICO

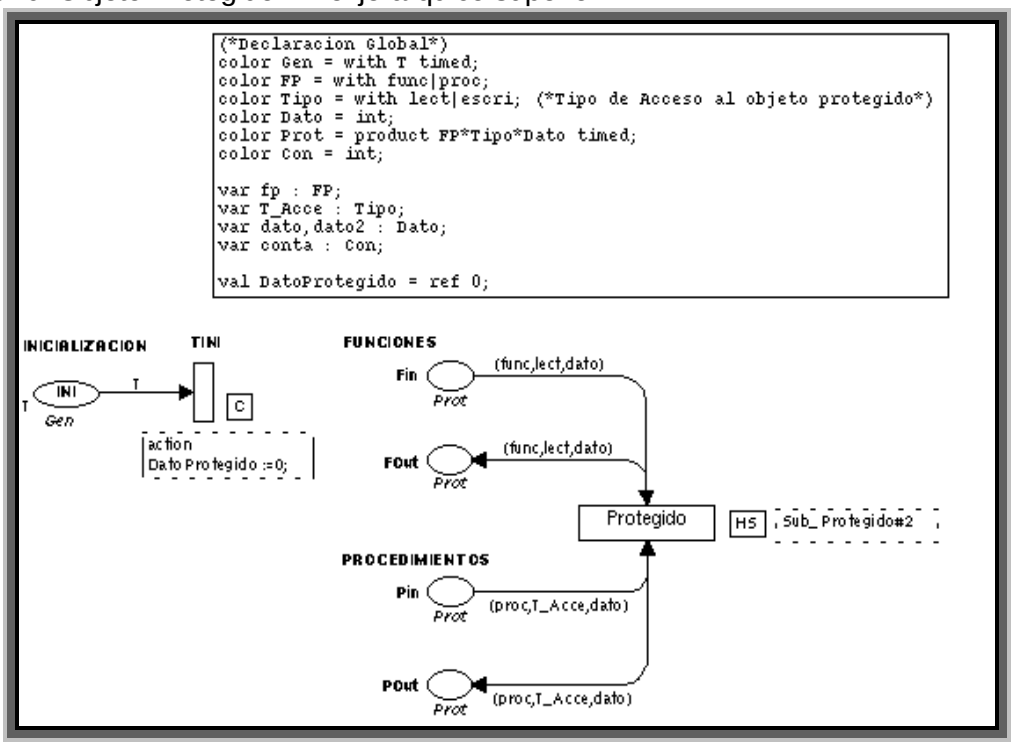
El objeto protegido presentado de esta manera, es útil para tareas que requieran acceder a datos protegidos, en el cual se establecen unas condiciones muy especificas. Se puede acceder a lectura del dato que protegen por medio de funciones o procedimientos, mientras que para escritura, se accede por medio de procedimientos únicamente. La representación en HRT-HOOD se presenta en la figura 39.

Figura 39. Objeto Protegido en HRT-HOOD



Este objeto es muy utilizado en sistemas distribuidos, para evitar que se escriban datos, mientras se encuentra en ejecución un proceso de lectura, o en el caso contrario. La escritura de los datos se hace solo una a la vez, mientras la lectura se puede efectuar de manera concurrente. En la figura 40 se presenta el primer nivel jerárquico del objeto protegido.

Figura 40. Objeto Protegido: nivel jerárquico superior

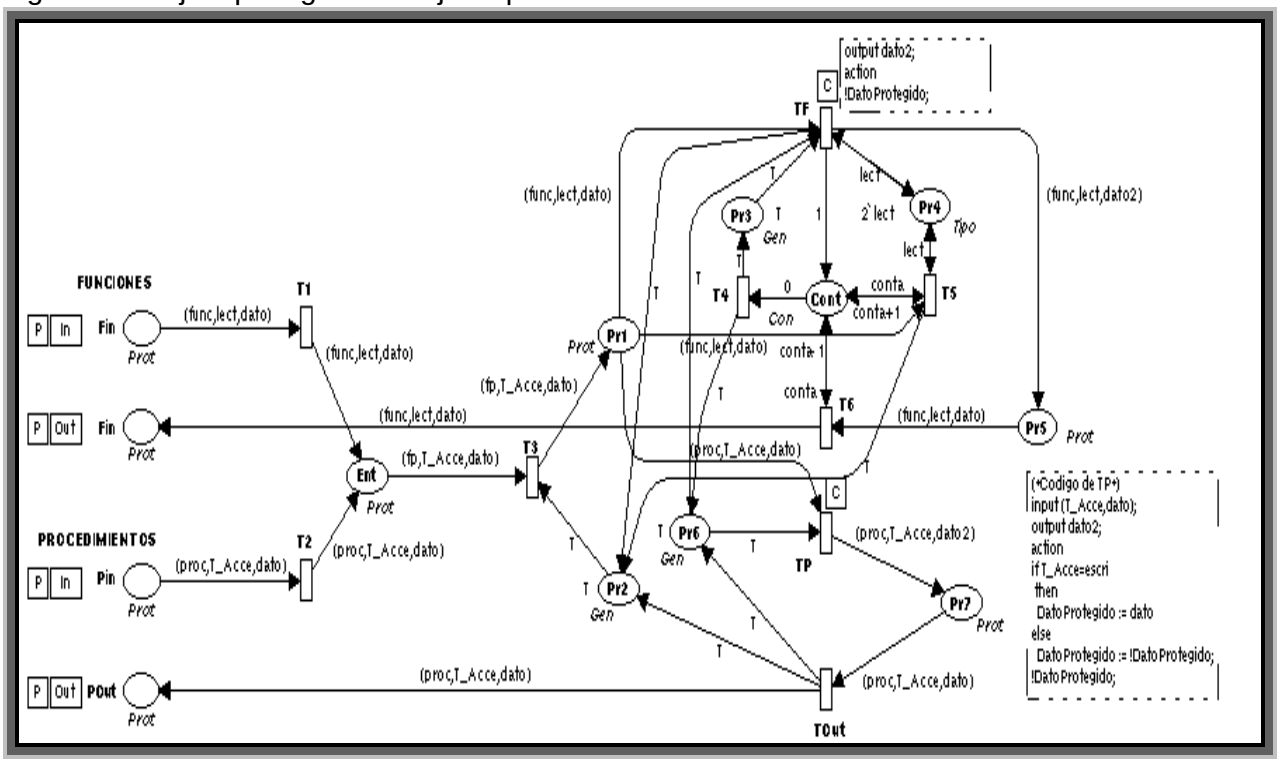


El objeto ofrece una interfaz de entradas y salidas: los lugares *Fin* y *Fout* permiten el acceso de las funciones al objeto, tanto como para entrada como para salida, respectivamente; *Pin* y *Pout* cumplen la misma función para los procedimientos.

El dato protegido se almacena en la variable referenciada *DatoProtegido*. Si se desea, el valor almacenado puede ser inicializado.

El segundo nivel se presenta en la figura 41. Los lugares de entrada *Fin* y *Pin*, activan a las transiciones *T1* y *T2*, para permitir el acceso de las solicitudes al buffer de entrada *Ent* (Entrada a dato protegido).

Figura 41. Objeto protegido: nivel jerárquico inferior



Cuando hay alguna solicitud en el lugar *Ent* y el objeto está disponible para aceptarla (esta condición se evalúa por medio del lugar *Pr2*), la transición *T3* se activa, y ubica la solicitud en el lugar *Pr1*, en donde se determina si la solicitud es una función ó un procedimiento. Si es una función, la transición *TF* (Transition Función) se activa, y en caso contrario, la transición *TP* (Transition Procedimiento) es quien lo hace. Para el primer caso, el disparo de la transición *TF* ocasiona que sea removido el token de *Pr2* (además de la marca de *Pr1*), para no permitir el acceso de más solicitudes mientras se realiza el proceso, y se remueve también el token de *Pr3* para indicar que el objeto está ejecutando una (ó más) solicitudes de funciones. Con la activación de *TF* se realiza el conteo de la primera función que utiliza al objeto, en *Cont* (Contador), y también se realiza el proceso de lectura del dato para ser entregado al lugar Pr5. Cuando aparece el primer token en *Cont* (recepción de la primera función) es posible atender a alguna otra tarea que haya llegado a *Pr1* de manera concurrente, por medio de *T5*, realizando el respectivo conteo. Cuando las funciones se van ejecutando, el token presente en *Cont*, se va decrementando, hasta que finalmente llega a cero, y el objeto es liberado con el disparo de *T4* (se colocan marcas en *Pr6* y en *Pr3*).

De otro lado, los procedimientos son más fácilmente atendidos por el objeto, debido a que solo no se presentan casos de concurrencia.

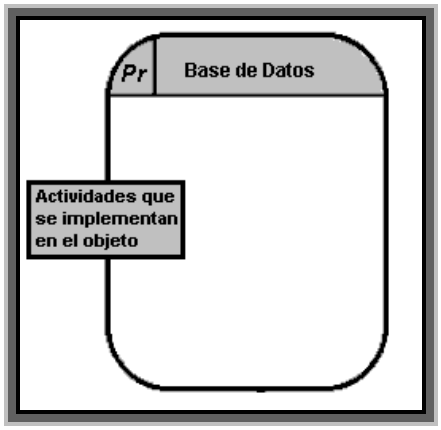
El formato de los tokens utilizados en este modelo es el siguiente:

(fp,T_Acce,dato) : fp (función/procedimiento) es el identificador del mensaje, T_Acceso indica si el acceso al objeto es de escritura o de lectura, y dato almacena el valor leído del objeto, ó en el caso de la escritura por medio de procedimientos, este campo indica el valor que será escrito en la variable protegida. Este formato puede ser alterado, si se utiliza este objeto para algo más que para almacenar una variable.

6.9 BASE DE DATOS SIMPLE

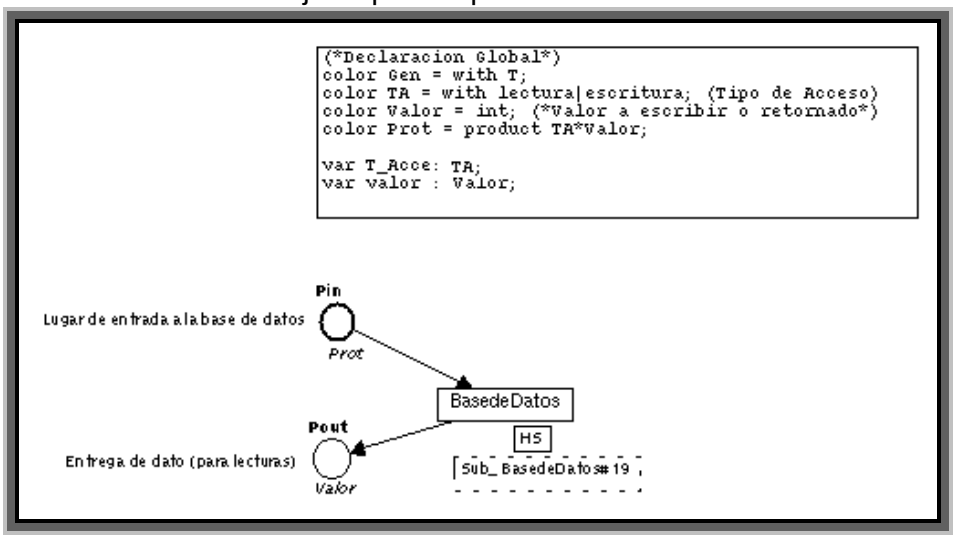
Este objeto permite proteger un dato para que no pueda ser accesado de manera simultánea por más de una solicitud. Su representación en HRT-HOOD se puede observar en la figura 42.La interfaz puede definir las tareas que deben ser sincronizadas.

Figura 42. Objeto Protegido en HRT-HOOD



El modelo de la base de datos, se hace como una simplificación del objeto protegido básico, para permitir el acceso a un dato. Este dato no podrá ser accesado de manera simultánea ni para su lectura, ni para su escritura. En la figura 43 se presenta el modelo de este objeto en redes de Petri.

Figura 43. Base de Datos: nivel jerárquico superior

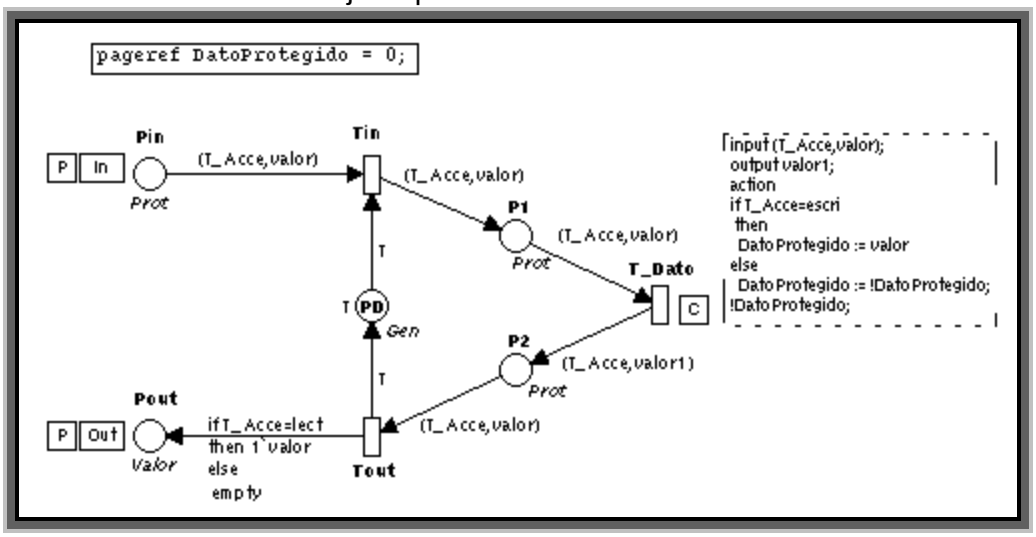


La base de datos puede ser accesada por el lugar *Pin*. La marca que llegue e *Pin* debe contener información sobre el tipo de acceso que realizará a la base de datos (lectura o escritura) y sobre el valor a leer o a escribir. Esto se observa en la declaración, donde dice que el token tiene dos campos: Un campo que está asociado al Tipo de Acceso *color TA*, y otro que es un Valor de *color int*.

Si el acceso a la base de datos es de escritura, el segundo campo de la marca será tomada en cuenta como el valor que reemplazará al dato presente en el momento de este acceso. Para el acceso de lectura el segundo campo no es importante del token; el valor de la lectura de datos será ubicado en el lugar *Pout*. En este caso, el dato almacenado en el objeto es de tipo entero, pero puede ser del valor que se necesite, solo se debe modificar el color *Valor* en la declaración.

La figura 44 contiene la base de datos a su segundo nivel de jerarquía.

Figura 44. Base de Datos: Nivel jerárquico inferior



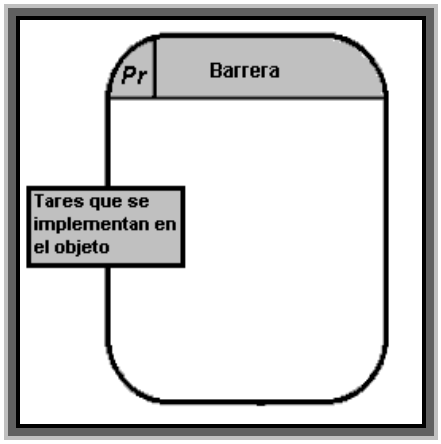
La transición *Tin* permite el acceso a la base de datos si el objeto no está ocupado, verificando esto por medio del lugar *PD*. Cuando *Tin* se dispara, produce un token en el lugar *P1* y remueve la marca de *PD*. Con base a la información sobre el tipo de acceso, la transición *T_Dato* determina si se reemplaza el valor protegido o si simplemente será entregado el valor del mismo al lugar de salida *P2* (el objeto fue diseñado para entregar siempre el valor del dato protegido). Después de este paso, la transición *Tout* queda activada, para que con su disparo, coloque una marca en *TD*, y si el tipo de acceso es de lectura coloque una marca con el valor del dato protegido en *Pout* (si el tipo de acceso es de escritura, no se colocará marca en *Pout*).

El dato protegido en el modelo es una variable (local a la pagina) tipo puntero, que es accesada por medio del segmento de código de la transición *T_Dato*.

8.11. Barrera

Este objeto sincronizador es muy útil cuando el sistema es critico en tiempo real, y cuando reúne varios subprocesos que pueden comunicarse entre ellos. La figura 45 es la representación del objeto en HRT-HOOD.

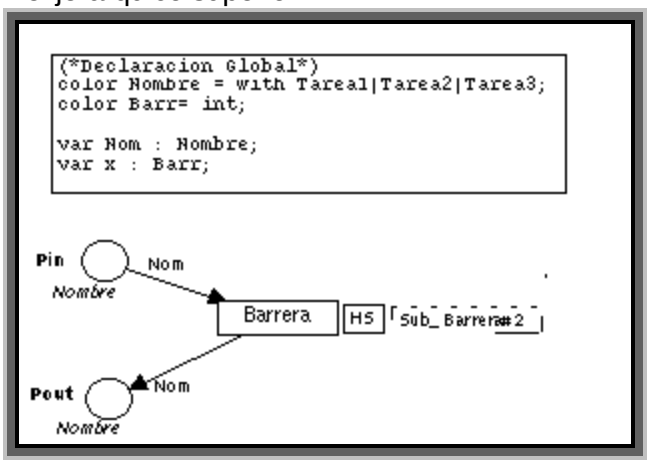
Figura 45. Barrera en HRT-HOOD



La barrera permite reducir los desfasamientos iniciales entre las tareas del sistema al detenerlas una a una hasta que llega la ultima tarea, para proceder a su liberación. Se utiliza en muchas aplicaciones para sincronizar la Inicialización, pero este componente puede servir en otros casos, como por ejemplo, el refresco sincronizado de varios datos en un monitor.

Su modelo en CP-nets a nivel superior, se presenta en la figura 46. En este modelo, las tareas que llegan a la barrera son recibidas por el lugar *Pin*, y cuando la sincronización es realizada, las tareas salen por el lugar *Pout*. En la declaración del modelo se han definido tres tareas como ejemplo, identificadas por el color *Nombre* (Tarea1, Tarea2 y Tarea3).

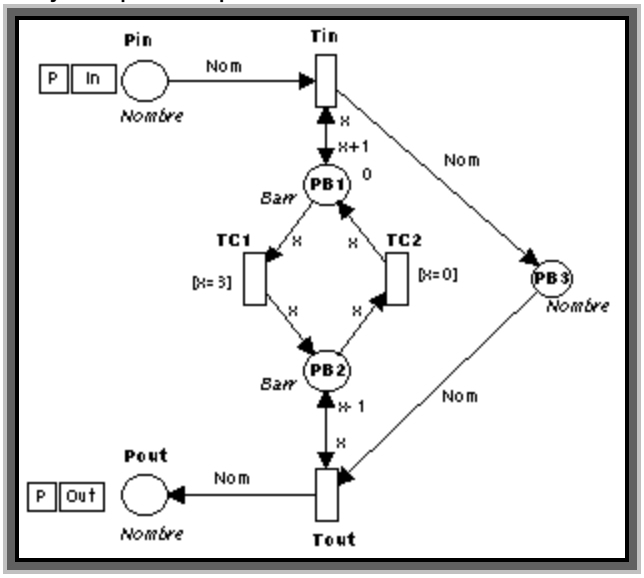
Figura 46. Barrera: nivel jerárquico superior



La figura 47 es la visión interna de la barrera. Cuando las tareas van llegando al objeto, son aceptadas por la transición *Tin*, la cual almacena el identificador de la tarea que ha llegado en el lugar *PB3*(Place Barrera). El lugar *PB1* contiene una marca que es de color entero (int), e inicialmente se encuentra con un valor de cero. Cuando una tarea llega a la barrera la transición *Tin* se activa, y el token de *PB1* se incrementa por medio los arcos entre estos dos elementos, llevando así un conteo de las tareas que han ingresado a la barrera.

El valor de la marca contenida en *PB1* está continuamente siendo comparado con una constante que representa el número de tareas que serán sincronizadas; esta comparación se realiza en la transición *TC1* (Transition Comparadora), por medio de un guarda, que como ejemplo en la figura (figura 47) tiene un valor de 3, es decir que se activará cuando la marca de *PB1* tenga un valor de 3 (tareas recibidas). Este número puede ser modificado para la aplicación.

Figura 47. Barrera: nivel jerárquico superior

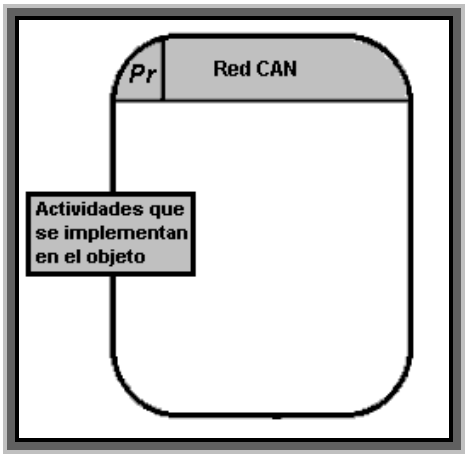


Cuando todas las tareas han llegado a la barrera, la transición *TC1* se activará, la cual con su ocurrencia removerá la marca del lugar PB1 y generará una marca en el lugar PB2 con el valor que contenía *PB1*. Con la marca presente en *PB2*, se activará la transición *Tout*, y de esta forma se dará paso a la liberación de las tareas. Cada vez que una tarea es liberada, el valor de la marca de *PB2* se decrementará hasta llegar a 0 (cero), lo que significa que todas las tareas han sido liberadas. Cuando el token es 0, la transición *TC2* eliminará la marca de *PB2* y colocará una marca con el valor de 0 al lugar *PB1*, quedando disponible para una nueva sincronización si es necesaria.

6.10 RED CAN

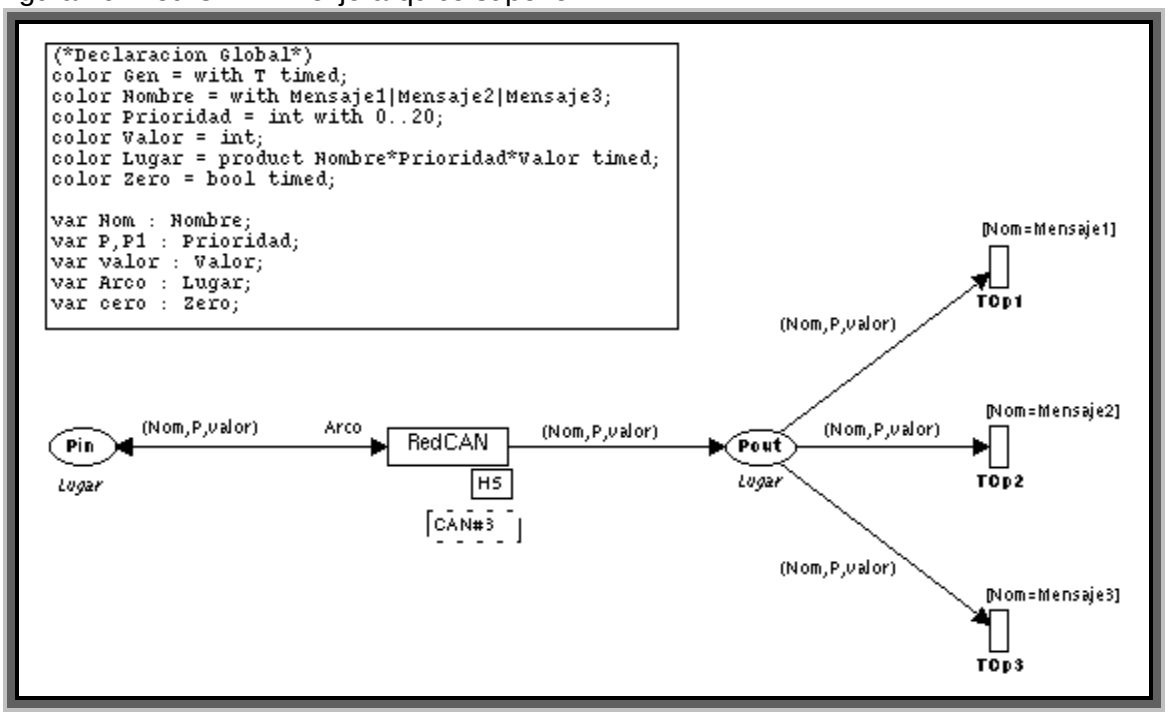
La red es un factor importante a tener en cuenta cuando se modelan sistemas en ambientes distribuidos con énfasis en tiempo real, debido a que esta puede retrasar los mensajes en la disputa por el medio, o en los retrasos propios de la Red. Se seleccionó la red CAN por sus ventajas operacionales en sistemas de tiempo real, y porque es un protocolo fácilmente implementable en redes industriales a pequeña y mediana escala. En la figura 48, se ilustra la representación en HRT-HOOD. Se modela como objeto protegido, ya que implementa esas características para los mensajes que toman el medio de transmisión.

Figura 48. Red CAN en HRT-HOOD



El modelo de la red CAN en el primer nivel jerárquico se presenta en la figura 49. Los mensajes que van a ser transmitidos a través de la red entran a este objeto por medio del lugar *Pin*, y salen por el lugar *Pout*. Los componentes en este nivel son solo *Po*ut, pin y la transición *RedCan*, las otras transiciones, simplemente representan la manera como los datos de salida deben ser identificados por cada uno de los receptores (son la implementación de los filtros que el chip CAN de cada nodo utiliza).

Figura 49. Red CAN: nivel jerárquico superior



El formato del mensaje (los mensajes son representados por las marcas) utilizado para el modelamiento es el siguiente:

(Nom,P,valor) : Nom (Nombre) representa el identificador del mensaje con efectos de visualización en la representación, *P* (Prioridad)indica la prioridad del mensaje y a la vez es el identificador real en la trama de CAN, *valor* representa a los datos que transporta la trama.

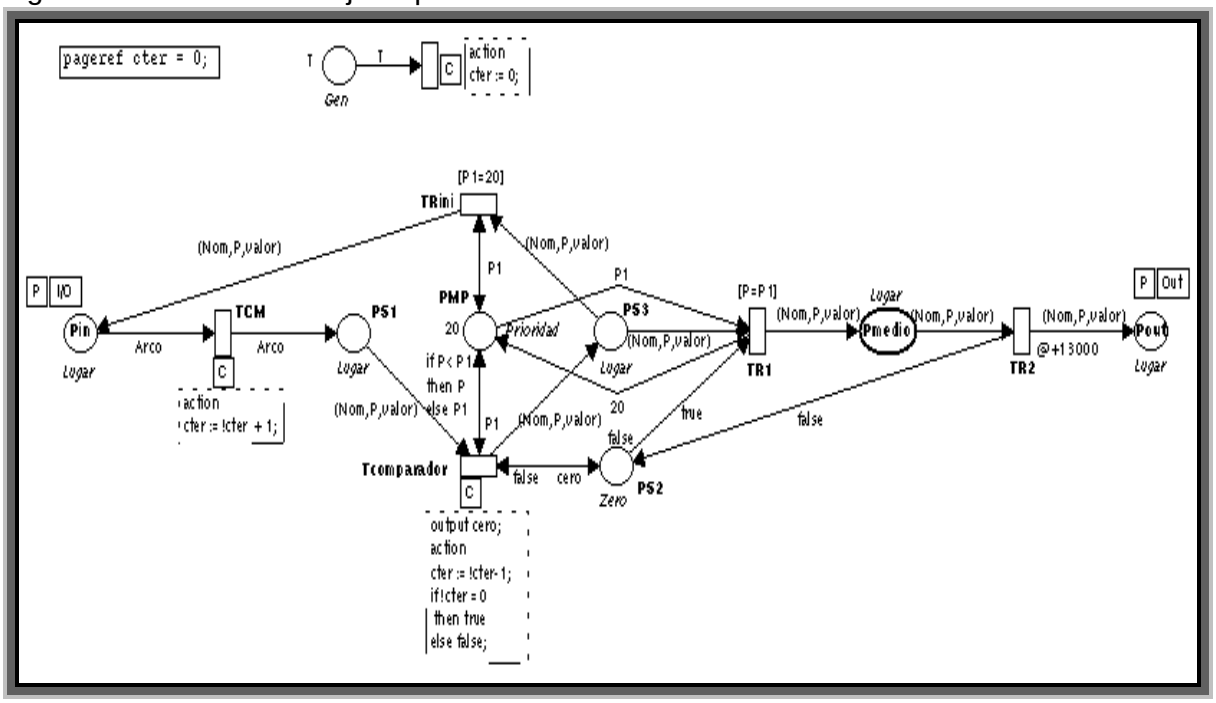
Una característica del protocolo CAN es que los identificadores son asignados a los mensajes, y no a los nodos como se realiza en Ethernet, por ejemplo. De esta manera, cada mensaje que se vaya a enviar por la red debe tener un identificador diferente.

Con respecto a las prioridades se destaca que entre menor sea el valor en este campo, mayor será la prioridad del mensaje.

En la figura 50 se presenta el modelo interno o nivel segundo en la jerarquía de la representación de la red CAN.

El modelo representa la manera como los mensajes acceden a la red. Se pueden observar los casos de disputa y su solución. Se observa también la ocupación del medio por el mensaje que se encuentre en el proceso de transmisión y la forma como se protege.

Figura 50. Red CAN: nivel jerárquico inferior



Para el correcto funcionamiento del modelo, se debe realizar la Inicialización del puntero (*cter*) que cuenta los mensajes que requieren ser transmitidos. Esto se debe realizar para no tener problemas durante la simulación (el puntero retiene el valor hasta que el PC es reiniciado). Esta Inicialización se puede realizar en algún otro objeto del sistema.

Cuando los mensajes entran al objeto, a través de *Pin* se encuentran con la transición *TCM* (Transition Conteo de Mensajes) que se encarga de contarlos para un posterior proceso de comparación; este conteo se realiza por medio de la variable referenciada a la pagina *cter*, esta variable debe ser inicializada en algún objeto (se inicializa en cero). Los mensajes ubicados en *PS1* (PlaceS) activan la transición *Tcomparador*, la cual decrementa a la variable referenciada *cter* hasta que sea 0, para asegurar que todas los mensajes que lleguen en un mismo instante de tiempo sean comparados, antes de realizar la asignación del medio. La prioridad de cada mensaje que activa a la transición

Tcomparador es comparada con el valor de la marca presente en *PMP* (Place Mayor Prioridad), por medio de los arcos que relacionan a *Tcomparador* con *PMP*, para determinar si la marca del lugar *PMP* debe ser reemplazada, con el fin de que este lugar contenga siempre el valor de la prioridad más alta. En otras palabras, se busca el numero más pequeño del campo de prioridad (que para el protocolo CAN es la prioridad más alta) entre los diferentes mensajes que se encuentran compitiendo por transmitir en un instante de tiempo dado, y se asigna al lugar *PMP*. Después de que todos los mensajes han sido comparados se encuentran almacenados en el lugar *PS3*, y por otra parte, *Tcomparador* cambia el valor de la marca del lugar *PS2* (por medio del segmento de código al cumplirse la condición de que *cter* es 0) de *false* a *true*, para ordenar a la transición *TR1* (Transition Red) que seleccione el mensaje que va a transmitir la red. La transición *TR1* utiliza al lugar *PMP* para determinar cual de los mensajes es el más prioritario, lo cual se realiza como se describe a continuación: El guarda, para poder activar la transición *TR1* necesita de un mensaje que tenga el valor de P igual al valor de la marca contenida en *PMP* (relacionada en el arco como P1), el valor que cumple esa condición es el más prioritario, ya que *PMP* después de realizar todas las comparaciones contiene el valor más prioritario de todos los mensajes.

Luego de que *TR1* ha seleccionado el mensaje más prioritario, éste se envía a la red con el disparo de la transición, colocando un token en *Pmedio*. El disparo de *TR1* ocasiona también que el token que había en *PS2* desaparezca, de esta manera, no se realizarán más comparaciones hasta que el medio sea liberado (protege el mensaje que se está transmitiendo). Finalmente, la ocurrencia de *TR2* remueve el token de *PMP* y le reasigna una marca con el valor que inicialmente contenía (correspondiente a un valor de prioridad menor a la mínima prioridad encontrada en los mensajes, o un número muy grande).

Cuando llega el valor inicial a *PMP* se retornan los mensajes almacenados en *PS1* al lugar de entrada (*Pin*) a través de *TRini*.

Con una marca en *Pmedio* la transición *TR2* se activa, y en su ocurrencia remueve el token de *Pmedio* y genera uno igual en *Pout* adicionándole un retardo correspondiente al tiempo que se tarda el mensaje en la red. Además, se genera también un token en *PS2* con el mismo retardo del token de *Pout* para seguir protegiendo al mensaje en la red; cuando el tiempo del retardo se agota, significa que el mensaje ya se transmitió, y se permite entonces realizar más comparaciones.

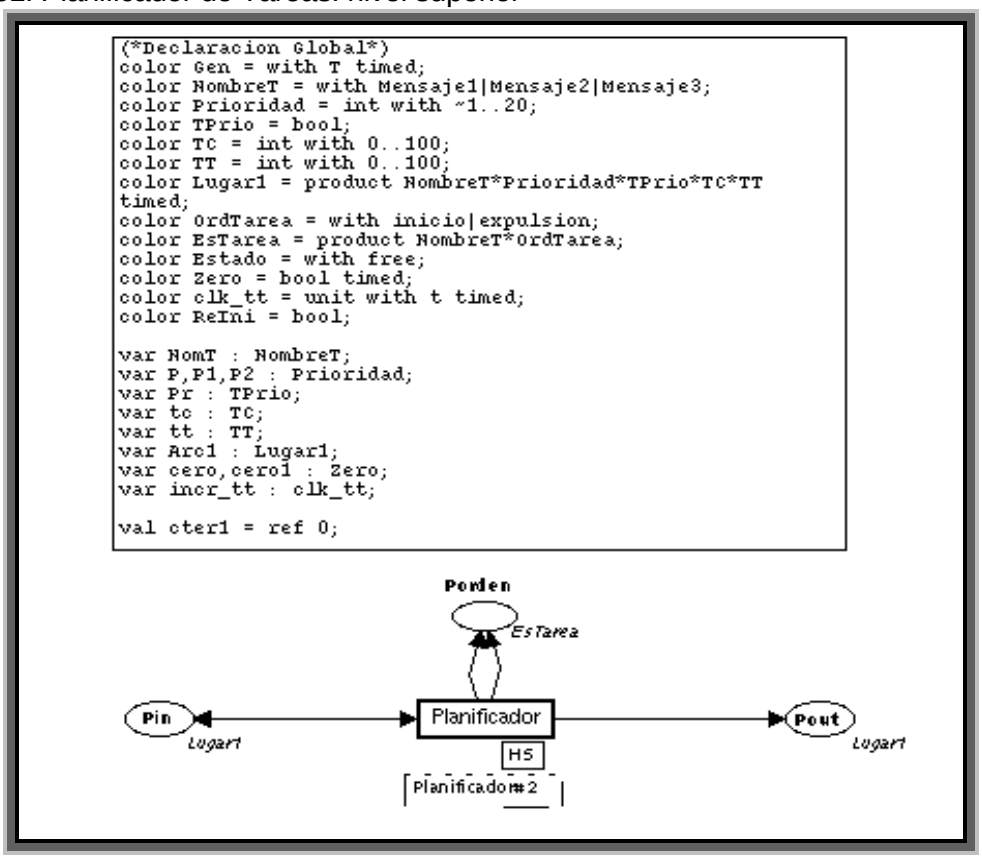
6.11 PLANIFICADOR DE TAREAS

Otra aplicación que se ha modelado en redes CP-nets es el planificador de tareas, que sirve para realizar validaciones del sistema planificado de manera dinámica. Este modelo comprende la planificación de tareas por medio de prioridades estáticas. La tarea realizada por el planificador de tareas es transparente para un programador, pero para un diseñador tiene especial importancia, sobre todo para los sistemas que se tratan en esta

tesis; en vista de que los métodos de planificación matemáticos, presentan inconvenientes cuando se cumplen algunas de las condiciones expuestas en secciones anteriores.

La figura 51, modela el nivel jerárquico superior del planificador de tareas. El planificador tiene un lugar de entrada de tareas *Pin*, otro de salida de tareas (para cuando finalizan la ejecución) *Pout* y un lugar para hacerle indicaciones a las tareas *Porden*.

Figura 51. Planificador de Tareas: nivel superior



Anteriormente se dio una idea superficial del formato de los tokens que actúan en las operaciones del planificador y en la interfaz de control de planificación de los objetos expropiables. Ahora se puede especificar mejor el formato de estos tokens teniendo en cuenta la declaración ilustrada en la figura 51:

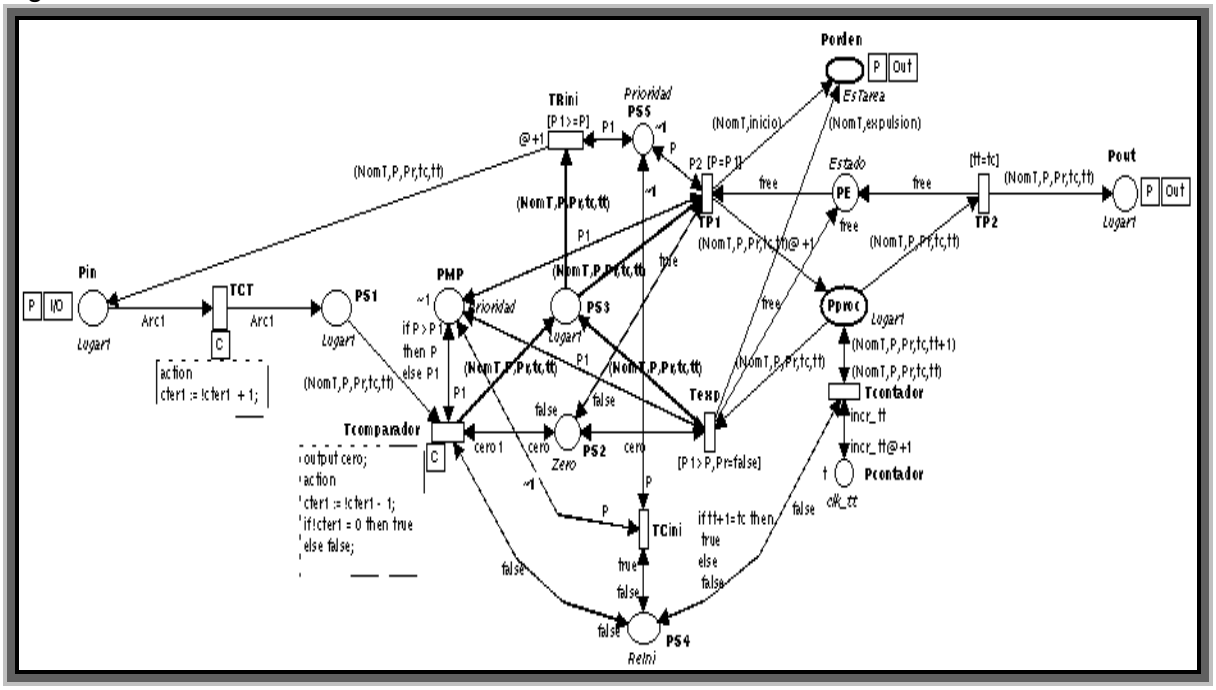
(NomT,P,Pr,tc,tt) : NomT representa el identificador de la tarea, *P* la prioridad de la tarea, *Pr* indica si una tarea es protegida o no (true = Protegida), *tc* es el tiempo de computo de la tarea y *tt* es el tiempo transcurrido en la ejecución de la tarea.

(NomT,est) : est representa el tipo de orden que el planificador le envía a las tareas, bien sea para indicarles que pueden iniciar sus actividades o que son expropiadas del procesador.

La prioridad máxima de las tareas en un sistema planificable corresponde al valor más grande en el campo P del token.

El tiempo de computo depende exclusivamente del tiempo que se demoran en ejecutarse todas las actividades. Es un poco difícil de obtener si no se cuenta con herramientas software que permitan conocer dicho tiempo, o en su defecto se debe conocer el tiempo de ejecución de las instrucciones incluidas en el código implementado, y hacer los cálculos correspondientes.

Figura 52. Planificador de Tareas: nivel inferior



Internamente el modelo del planificador se ilustra en la figura 52. La estructura de comparación y selección de prioridades mayores es muy similar a la del modelo realizado para le red CAN, pero se invierten las comparaciones para saber cual tarea es la más prioritaria, porque en CAN el valor más pequeño en el campo de prioridades definía el mensaje más importante, mientras que con el planificador, el valor mayor en el campo de prioridades corresponde a la tarea más importante.

Otras características importantes de este modelo son que tiene implementado un mecanismo de expulsión de tareas y un sistema de temporización especial para la aplicación.

Las tareas son recibidas por el lugar *Pin* y contadas por la transición *TCT* (Transition Conteo de Tareas) sobre la variable referenciada *cter1*, la cual debe haber sido inicializada en algún otro objeto. Cuando las tareas están en *PS1* se realiza el mismo proceso de comparación que para el modelo de CAN, con el fin de que cuando las comparaciones culminen, el lugar *PMP* contenga una marca con el valor que indica cual es la tarea más prioritaria. Una vez terminado este proceso de comparaciones, las tareas se encuentran en *PS3* y el valor de la marca de *PS2* cambia e false a true. Con las anteriores condiciones, la transición *TP1* se habilita y con la ayuda de *PMP* determina cual es la tarea más prioritaria, para asignarla al procesador. El disparo de *TP1* tiene unos efectos un poco distintos a los ocasionados por el disparo de la transición *TR1* en el modelo de la red CAN. Tras el disparo de *TP1*, al lugar *PS2* se le asigna un token de valor *false*, con lo que se permite seguir realizando comparaciones en *Tcomparador*, a la transición *PMP* se le retorna un valor igual al que tenia antes, es decir, el valor mas prioritario permanece en ese lugar, hasta que la tarea correspondiente a esa prioridad termine sus actividades, la ocurrencia de *TP1* cambia el valor de *PS5* (colocándole el valor de la tarea que accedió al procesador), con el fin de que se las tareas almacenadas en *PS3* puedan ser retornadas a *Pin* por medio de *TRini*. Las tareas retornadas a *Pin* son retardas (con la mínima unidad de tiempo) para que no queden en bucle infinito de comparaciones. El disparo de *TP1* remueve el token *free* del lugar *PE* (Place Estado del procesador), para indicar que una tarea tomó posesión del procesador, también crea un mensaje que le indica a la tarea a ejecutar, a través de *Porden* que ya puede iniciar sus actividades, y finalmente el disparo de *TP1* asigna la tarea al lugar que representa al procesador en acción, *Pproc* (Place procesador).

La expulsión de la tarea por la presencia de otras más prioritarias, se realiza por medio de la transición *Texp* . El guarda de esta transición evalúa si en *PMP* ha recibido un valor de prioridad mayor a la prioridad de la que se está ejecutando, y además pregunta si la tarea es protegida (campo Pr de la tarea que se está ejecutando), caso en el cual no importa si ha llegado una tarea más prioritaria, la tarea que está ocupando al procesador no será expulsada. En el caso de que *Texp* se active, con su disparo removerá el token que ocupa el lugar *Pproc* y lo envía** a *PS3*, se genera un token para *PE* en el que se le dice que el procesador queda libre, y se envía una orden de expulsión a la tarea que ha sido expropiada, en el lugar *Porden*. El arco entre *PS2* y *Texp*, se utiliza para que no se realice ninguna comparación hasta que la acción de expulsión sea concluida.

El conteo del tiempo transcurrido de la tarea se realiza por medio del lugar *Pcontador* y la transición *Tcontador* . Por cada activación de la transición se realiza un incremento de *tt* en la tarea que se encuentra en *Pproc*, y se genera un token retardado en una unidad de tiempo para el lugar *Pcontador*, y de esta manera, *Tcontador* solo se activará cuando ese tiempo haya transcurrido. Cuando *tt* es igual a *tc*, significa que la tarea ha sido concluida, y TP2 se activa para generar un token en el lugar de salida de tareas *Pout* y otro token en el lugar *PE* (para indicar que el procesador se encuentra libre).

El lugar *PS4* está asociado con la finalización de las tareas, y su función es la de reemplazar los valores de los lugares *PMP* y *PS5* por los valores inicialmente establecidos, los cuales corresponden al valor –1 (prioridad mínima). Por medio de los arcos que relacionan a *PS4* y a *Tcomparador* se bloquean las comparaciones, cuando la tarea que se estaba ejecutando termina su operación, mientras se realiza el proceso de Inicialización de los lugares mencionados (*PMP* y *PS5*).

7. CONTROL Y MONITOREO DE SISTEMA DISTRIBUIDO DE NIVEL (EJEMPLO DE APLICACIÓN)

Como ejemplo de aplicación de la estrategia de diseño de sistemas continuos de control industrial a ambientes distribuidos, se presenta un proyecto, el cual está siendo desarrollado actualmente por el grupo de investigación *Desarrollo Hardware y Software para Procesos de Ingeniería*, de la *Corporación Universitaria Autónoma de Occidente* (del cual formo parte). El sistema consiste en el control de nivel de un tanque con el lazo de realimentación pasando a través de una red CAN (es un NCS). Además de esto, se puede realizar monitoreo remoto del sistema, con posibilidad de hacer modificaciones del valor deseado de referencia (setpoint). Para realizar el análisis a nivel arquitectural, no se tendrán en cuenta perturbaciones suplementarias, o factores externos y ajenos al funcionamiento del proceso.

Para el desarrollo del proyecto se apoya en gran parte en el flujo de diseño planteado en la figura 3, permitiendo desarrollar las diferentes fases del diseño de manera paralela y a la vez tener una relación entre las diferentes etapas de diseño. Los detalles del modelamiento del sistema en HRT-HOOD están siendo tratados en otro proyecto de grado titulado *Flujo de Diseño Para el Desarrollo de aplicaciones en Tiempo Real Usando Lenguaje ADA*, además en ([19]) se detalla toda la metodología de HRT-HOOD y se presentan algunos ejemplos de aplicación. Los niveles siguientes al diseño arquitectural del sistema son implementados en la tesis presentada en ([20]), en la cual se modifica del flujo de diseño para aplicar las técnicas de Co-Diseño Hardware-Software.

En la primera parte de este capitulo se dará una breve descripción del sistema, para identificar algunas características generales importantes para el desarrollo del modelamiento. Después de esto, se describirán los elementos utilizados para la descripción del sistema, y en adelante se desarrollará la estrategia de diseño.

Las validaciones temporales y funcionales del sistema se realizan en la herramienta software Design/CPN. La estructura del sistema se diseña utilizando el *Editor* de la herramienta, el cual permite crear fácilmente redes de Petri Coloreadas, y permite realizar una descripción en jerarquías del sistema. Design/CPN detecta errores de sintaxis y estructurales, y da la ubicación de la página que los contiene, permitiendo de una manera relativamente fácil su corrección. Esta característica en parte representa la validación funcional del sistema.

[19] BURNS, Alan. HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems. 1995. 44 p.
[20] LIZCANO, Jorge y MACIAS, Hugo. Implementación de un módulo de sensado y de actuación para el control de nivel de un tanque mediante la técnica de codiseñoHardware, – Software. Cali, 2003, 122 p. Corporación Universitaria Autónoma de Occidente. Unidad académica eléctrica y electrónica.

Design/CPN permite además simular los modelos implementados, de esta forma se logra analizar al sistema evolucionando en el tiempo con el fin de realizar las validaciones temporales y funcionales. Los retardos implementados en las transiciones y los arcos, se verán reflejados en el campo temporal del simulador. Los desplazamientos de los tokens se observan gráficamente, y de esta manera es posible observar el comportamiento funcional del sistema. Adicionalmente a la simulación, Design/CPN puede generar los diagramas de alcanzabilidad del sistema, lo cual es un muy buen complemento para el análisis del sistema.

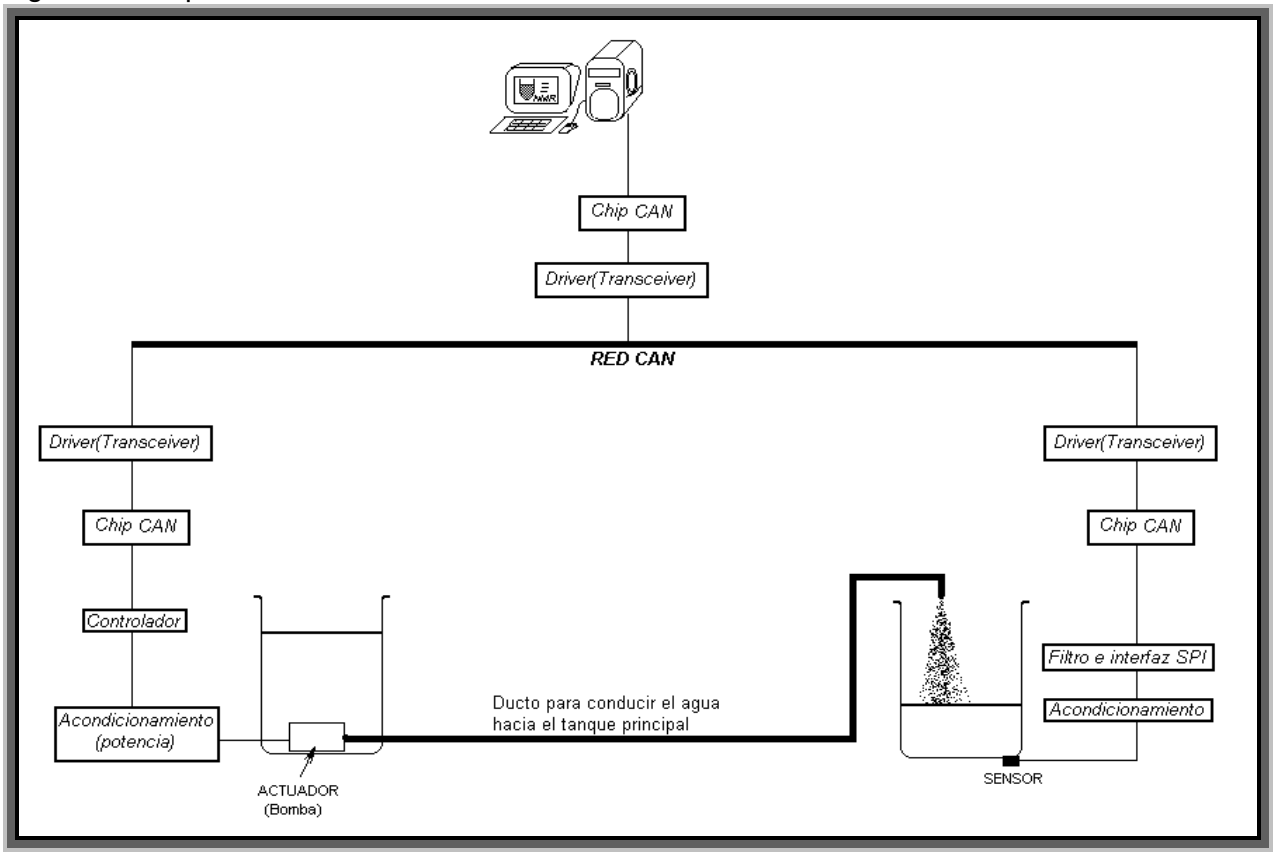
De aquí que se presente solo la estructura del modelo en esta tesis.

7.1 DESCRIPCIÓN DEL SISTEMA

Partiendo de las especificaciones del sistema, en la figura 53 se presenta un esquema que permite tener una idea global de la composición del sistema.

Como se puede observar en la figura, se van a realizar mediciones de nivel de un tanque en un sitio determinado, las cuales deben ser transportadas hacia dos destinos (ubicados en sitios diferentes); uno de ellos es el controlador de nivel, el cual extrae líquido desde un tanque y lo envía hacia el tanque al que se le realiza el control de nivel (por medio de una *moto-bomba*), y el otro es una terminal que puede servir para un computador, u otros elementos, que serán utilizados para el monitoreo del sistema e inclusive para realizar control manual por medio de la interfaz de usuario, convirtiéndose este terminal en un SCADA.

Figura 53. Esquema Global del Sistema



El tiempo real para el transporte de la señal de medición se maneja por medio del protocolo CAN, ya que este brinda, además de una velocidad relativamente alta, un control sobre los mensajes que pueden transitar por el medio de transmisión; esto último se debe al manejo de prioridades sobre los mensajes. El trabajo del protocolo CAN se centra especialmente en el nivel de enlace de datos del modelo de referencia OSI. Para implementar el protocolo, cada nodo que está interconectado a través de la red deberá tener un Chip CAN y un driver. El primero se encarga de cumplir con todos los requerimientos que el protocolo CAN formula (con este chip la implementación del protocolo CAN es casi transparente al nodo). La función del segundo elemento es adaptar físicamente las señales de los mensajes, para que puedan ser transmitidos por el medio de transmisión (conversión de voltajes a diferenciales, entre otras actividades).

Uno de los nodos se encarga de tomar lecturas periódicas de los valores de nivel. El sensor utilizado para hacer las mediciones de nivel es *ultrasónico* . Se implementa un filtro promediador para mejorar los valores de nivel. Luego de tomar dichas mediciones, se comunican al Chip CAN, por medio de una interfaz SPI. Este nodo cuenta también con una pequeña interfaz, compuesta por un teclado matricial y un LCD, que permite tener información del proceso en ese punto, y además por medio de esta interfaz se puede modificar la referencia (setpoint).

Por su parte el nodo del controlador recibe los valores de nivel y con base a ellos, calcula la acción de control respectiva, y se la aplica al actuador, elemento que también se encuentra en este nodo. La recepción se realiza de manera esporádica, con lo cual el periodo de actualización de la acción de control, depende exclusivamente de la periodicidad de la llegada de los valores de nivel; se puede decir que el período de esta actualización es el mismo del período de muestreo del nivel. Una ventaja de la captura esporádica, en este sistema, es que se estarán actualizando las acciones de control en tiempo real, sin inconvenientes (para realizar la captura de manera periódica se requiere de una sincronización entre el nodo que envía y el nodo que recibe, esta se implementa por medio de encuestas en el nodo controlador). Una vez calculada la acción de control, ésta será enviada a la red, para que el monitoreo remoto de este dato, también se pueda realizar (se envía el valor del porcentaje de apertura).

Para la parte de monitorización se utilizará un computador, el cual tendrá una interfaz de usuario soportada en el lenguaje de programación ADA. Este lenguaje se utiliza porque presenta características especiales en cuanto a la implementación y comportamiento manejo para sistemas en tiempo real. En el computador se podrán observar el estado del actuador , el nivel presente en el tanque y el setpoint; este ultimo podrá ser modificado desde la interfaz desarrollada. Esta parte del proyecto aún se está desarrollando en el trabajo de grado titulado *Flujo de diseño para el desarollo de aplicaciones en tiempo real usando lenguaje ada*.

7.2 COMPONENTES A UTILIZAR

Para modelar el sistema se hace uso de casi todos los componentes planteados, ya que el sistema implementa una gran variedad de actividades.

Entre los más utilizados se encuentran los siguientes componentes:

-Componentes aperiódicos: Implementan funciones de recepción y respuesta a eventos, como por ejemplo, cuando un dato de la red llega al nodo, por medio del uso de un objeto aperiódico se logra la captura del mensaje.

-Componentes periódicos: Utilizados para realizar procesos de monitorización de la red, o de lectura del sensor.

-Componentes Periódicos y Aperiódicos Expropiable: La expropiación fue utilizada en el modelamiento de las tareas implementadas sobre Ada.

-Red CAN: En la práctica la red utilizada, por los motivos expuestos anteriormente, fue la red CAN, y se utilizó su modelo para permitir analizar de los mensajes desde que son transmitidos hasta cuando son recibidos.

-Planificador: Dentro del proyecto, se realizó el estudio de las ventajas que ofrece un planificador de tareas con prioridades estáticas[21], entonces se seleccionó para trabajar junto con Ada al sistema operativo Linux, junto con su kernel de tiempo real, RTLinux. Por esta razón se implementó dicho planificador en el modelo, con el fin de analizar, los conflictos entre las tareas, retardos, etc.

-Funciones y procedimientos: Para representar las diferentes funciones implementadas en los diferentes objetos, además de las tareas, se utilizaron, las funciones y los procedimientos, para tareas comunes u operaciones específicas que no necesariamente fueron implementadas dentro del objeto.

También se implementaron diferentes arreglos para modelar tareas como las protegidas, o cuando se agruparon funcionalmente algunas actividades comunes a varios objetos. Para realizar las relaciones entre los componentes se utilizaron pequeños arreglos de transiciones y lugares, que son ilustrados claramente claros. Algunas acciones requieren de una implementación adicional que no es muy compleja, y se ve notablemente reducida por la utilización del lenguaje que soportan las redes de Petri Coloreadas en la herramienta de edición y simulación (Design/CPN).

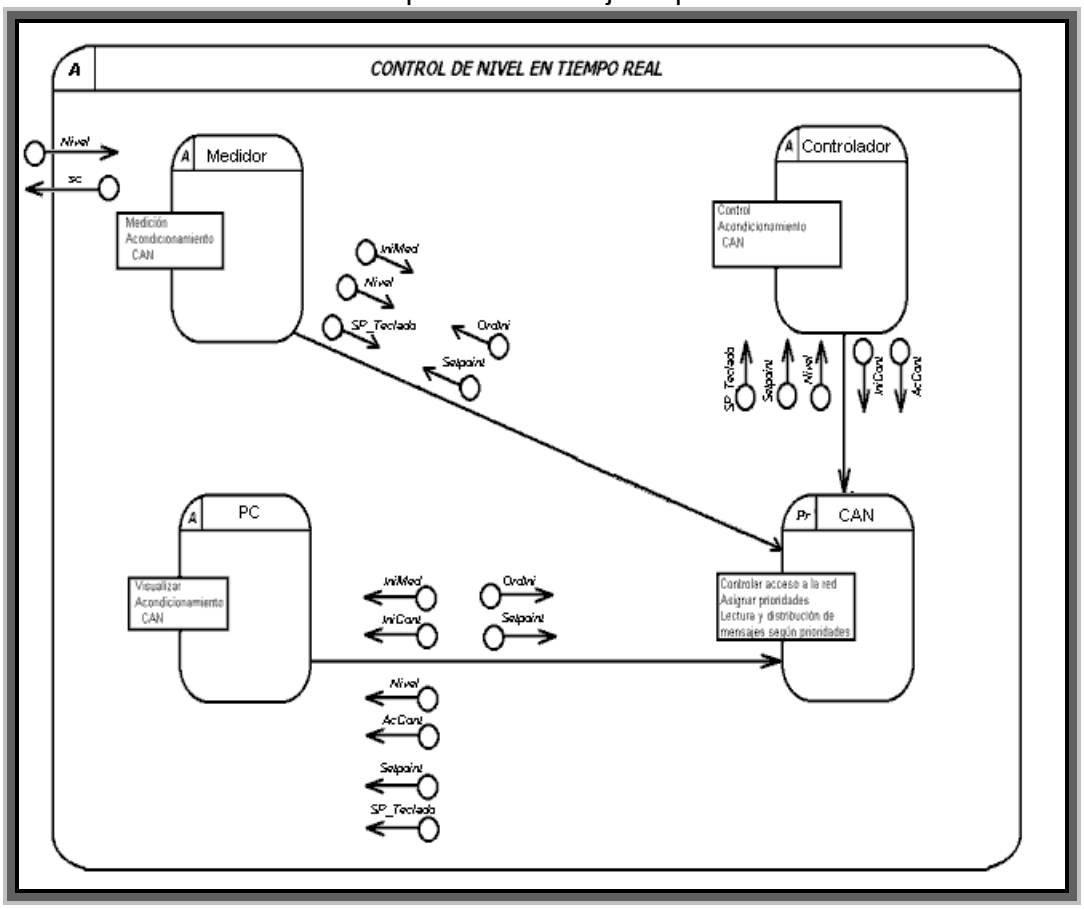
7.3 REPRESENTACIÓN DEL MODELO EN HRT-HOOD

Como primer paso se realiza el modelo en HRT-HOOD, para tener una idea de los elementos que integran el sistema. HRT-HOOD fue diseñado en especial partiendo del lenguaje de programación ADA, sin embargo, y como se podrá observar, no solo es útil para modelar software.

El primer nivel del modelo se presenta en la figura 54. Este nivel presenta cuatro componentes: *Medidor*, *Controlador*, *PC* y *CAN*.

[21] SALINAS, patricio. Tutorial UML [En línea]. [Santiago de Chile, Chile], octubre de 1996 [Citada en noviembre 2002]. Disponible en internet. <http://www.dcc.uchile.cl/~psalinas/uml/ >

Figura 54. Modelo en HRT-HOOD: primer nivel de jerarquía del sistema



-Objeto *medidor*: Su función es la de periódicamente tiempo tomar una medición de nivel en el tanque a controlar, y acondicionarla de tal forma que tenga un formato valido para ser llevada al objeto CAN. El objeto medidor se ha clasificado como activo porque en su interior contiene elementos que tienen actividad periódica, y pasiva.

-Objeto *controlador*: Recibe un valor de nivel actual, desde la Interfaz CAN y con base en él calcula una acción de control y de esta forma controlar la cantidad de líquido que entra al tanque.

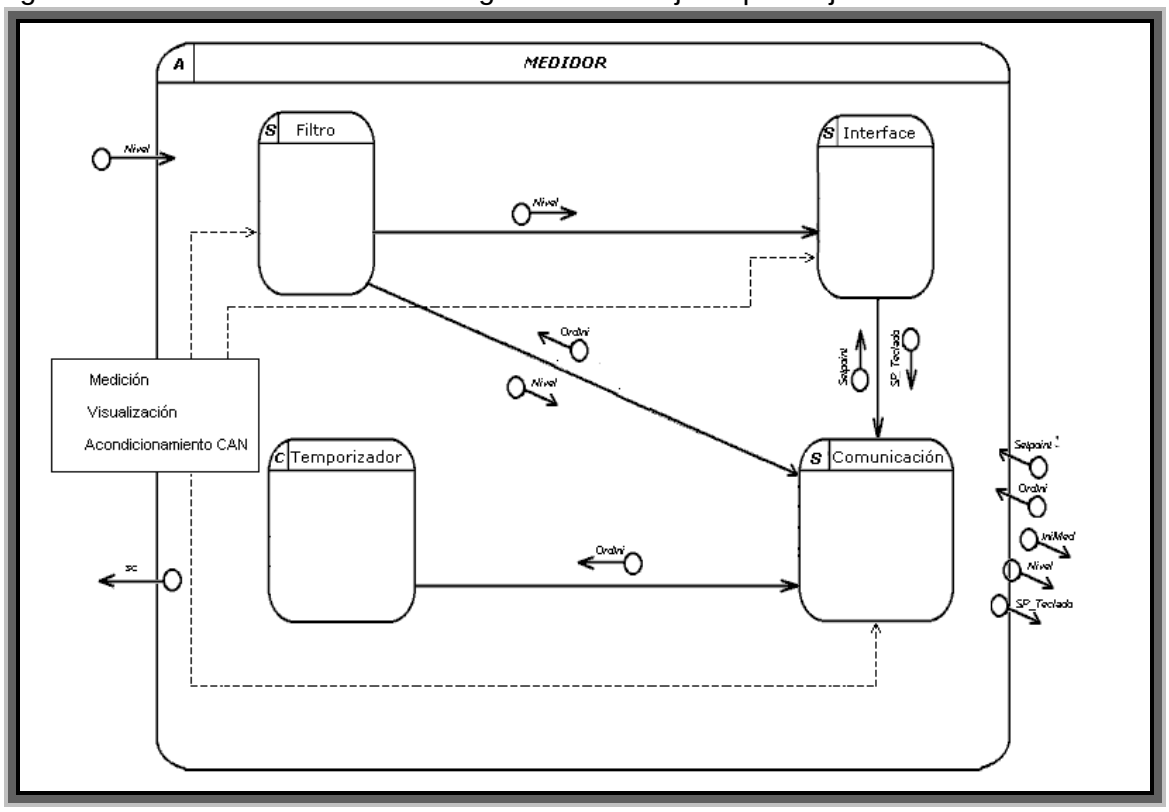
-Objeto *PC*: A este sitio llega información del estado del actuador, del nivel actual del tanque y de otro tipo de información, con el fin de tener reportes del comportamiento, o para realizar monitorización remota, de la planta en un computador.

-Objeto *CAN*: Representa las funciones del protocolo CAN y el medio de transmisión para que la información fluya desde un terminal hacia otro. Recibe información de nivel y de setpoint (SP_Teclado) desde el medidor, para enviársela al controlador y al PC, recibe

información del estado del actuador desde el controlador y se la enviará al PC, entre otros mensajes.

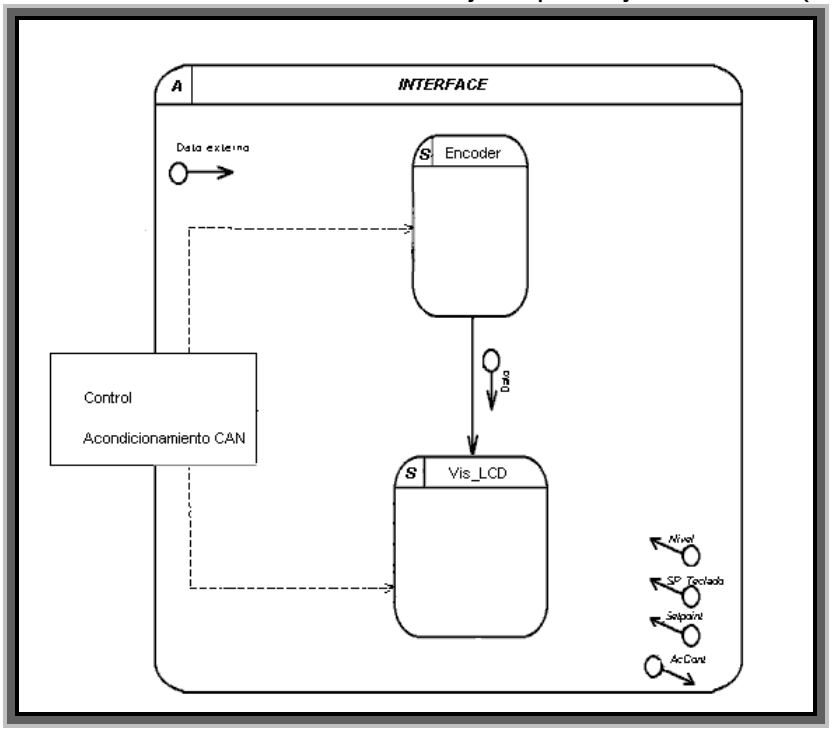
En las figuras restantes de esta sección, se presentarán los modelos siguientes en la jerarquía, para cada uno de los objetos del primer nivel. Su explicación se dará junto con las redes de Petri Coloreadas del sistema en la siguiente sección.

Figura 55. Modelo en HRT-HOOD: segundo nivel de jerarquía objeto *medidor*



La figura 55 ilustra el modelo para el objeto *medidor*. Se presentan cuatro objetos, de los cuales, se puede descomponer en otros niveles de jerarquía al objeto *interface*. La figura 56 muestra el tercer nivel jerárquico del sistema, para el *medidor* descomposición.

Figura 56. Modelo en HRT-HOOD: tercer nivel de jerarquía objeto *interface* (Medidor)



Las figuras 57 y 58, representan la descomposición de los dos niveles jerárquicos que contiene el objeto *controlador*. El único objeto que se descompone de *controlador*, es *control* el cual contiene dos objetos más para realizar su función.

Figura 57. Modelo en HRT-HOOD: segundo nivel de jerarquía objeto *controlador*

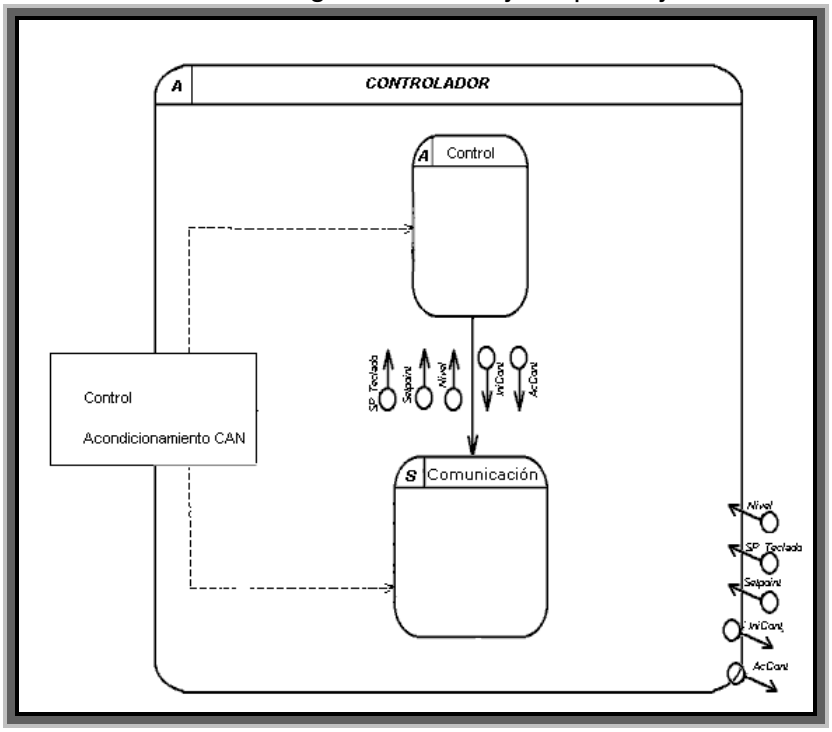
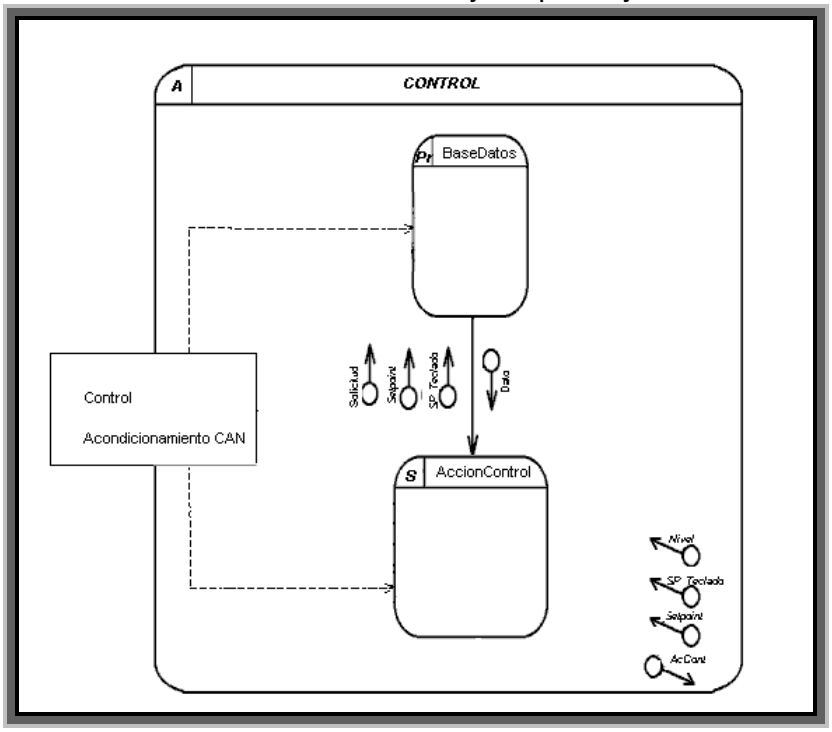


Figura 58. Modelo en HRT-HOOD: tercer nivel de jerarquía objeto *control*



En las figuras 59, 60 y 61 se presentan los niveles jerárquicos correspondientes al objeto *PC*. La figura 59 muestra al objeto *PC*, del cual se desprenden dos niveles más de jerarquía: *Procesamiento* (figura 60) e *Interfaz* (figura 61).

Figura 59. Modelo en HRT-HOOD: segundo nivel de jerarquía objeto *PC*

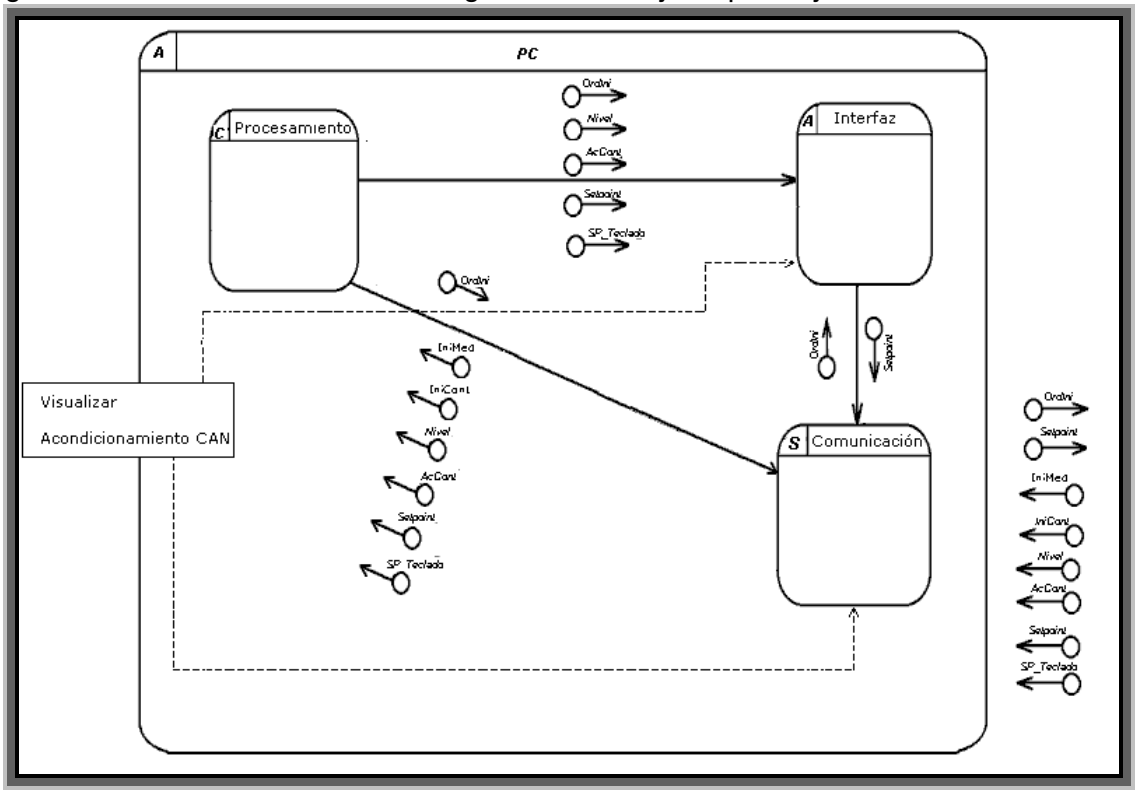


Figura 60. Modelo en HRT-HOOD: tercer nivel de jerarquía objeto *procesamiento*

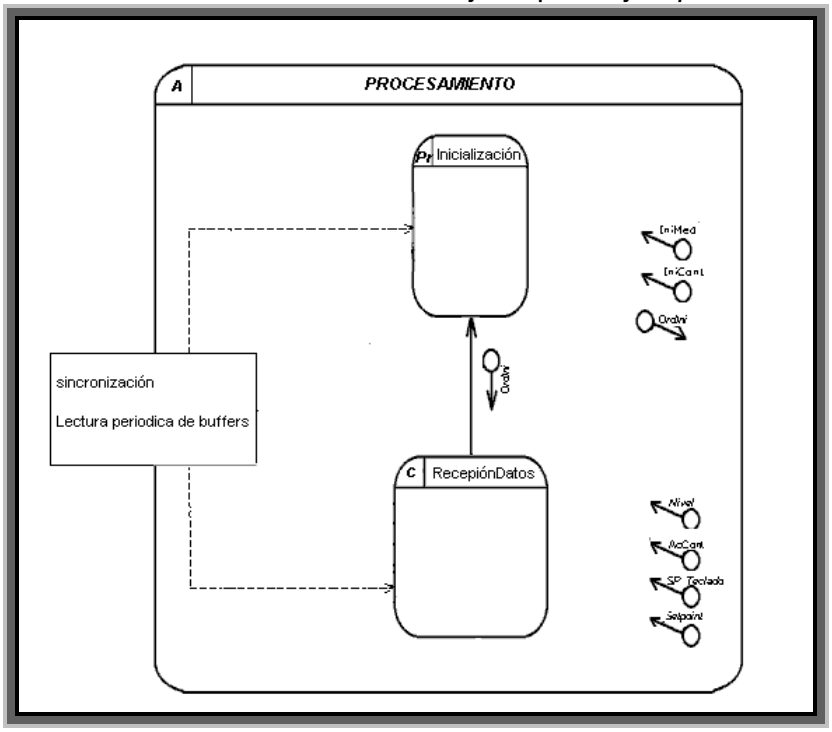
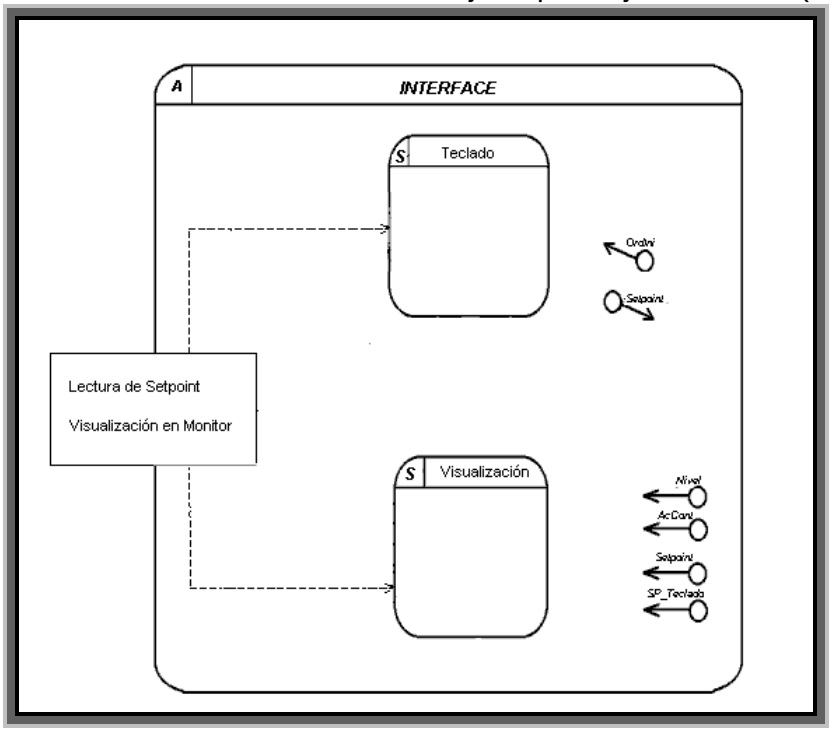


Figura 61. Modelo en HRT-HOOD: tercer nivel de jerarquía objeto *interface* (*PC*)



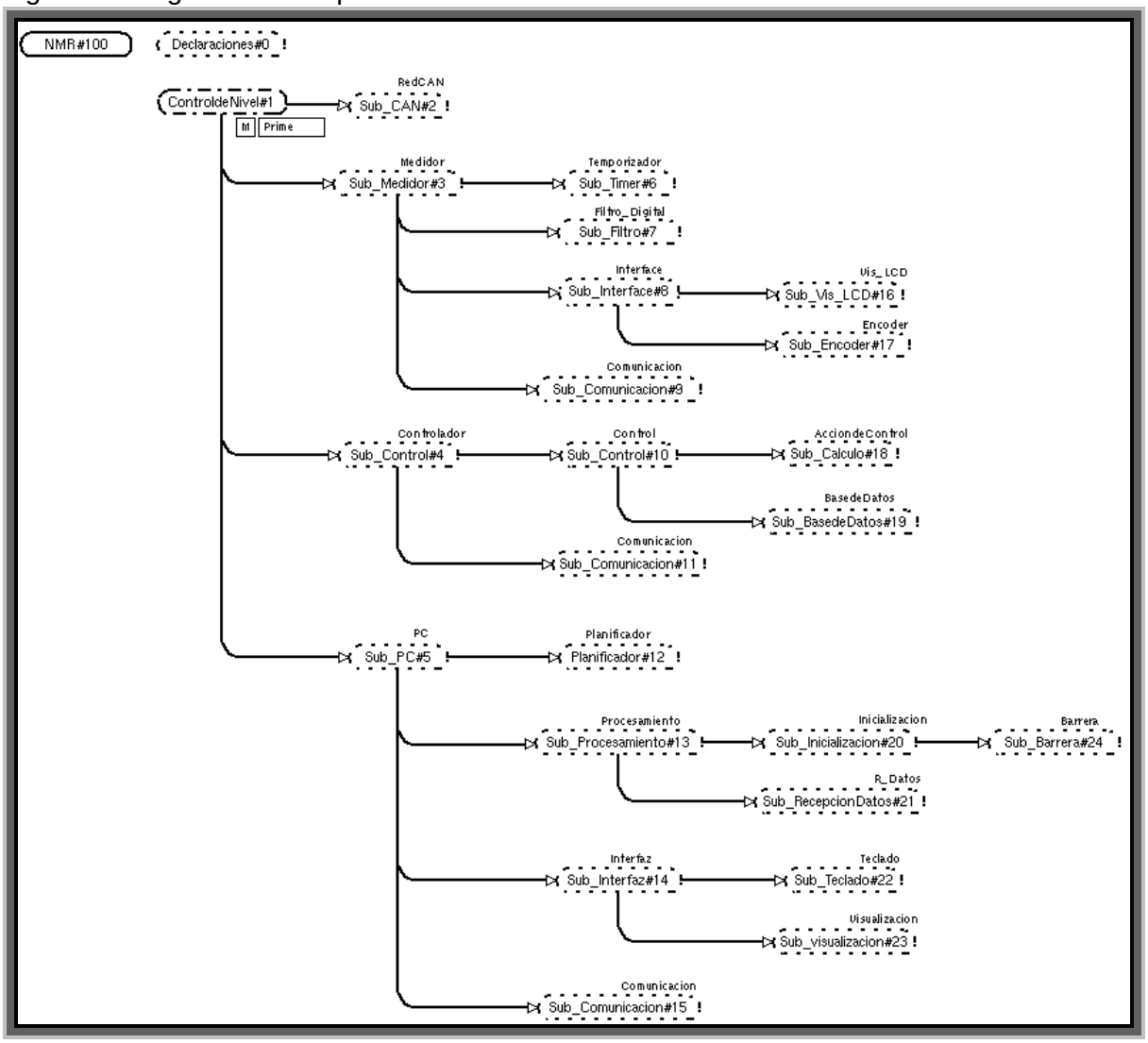
7.4 MODELO EN REDES DE PETRI COLOREADAS

Una característica importante de las redes de Petri es que representan sistemas de manera abstracta: no se especifican técnicas de diseño, solo se modela el funcionamiento del objeto o componente. En el sistema modelado se puede observar que este no está orientado hacia ningún lenguaje de programación, ni hacia técnicas de diseño especificas (cableado, compuertas, PLDs, PLCs), simplemente se representan la estructura del sistema, y se validan sus actividades.

Al igual que en HRT-HOOD, por medio de las redes de Petri Coloreadas el modelo del sistema se puede observar de manera jerárquica, de esta manera un sistema grande como el que se presentará enseguida puede resultar sencillo de entender. Puede realizarse la comparación con el modelo de HRT-HOOD, si se desea.

Para iniciar, se presentará la pagina de Jerarquía, que es donde se muestran las relaciones de las diferentes páginas diseñadas. La figura 62 muestra la página jerárquica del sistema control de nivel en red.

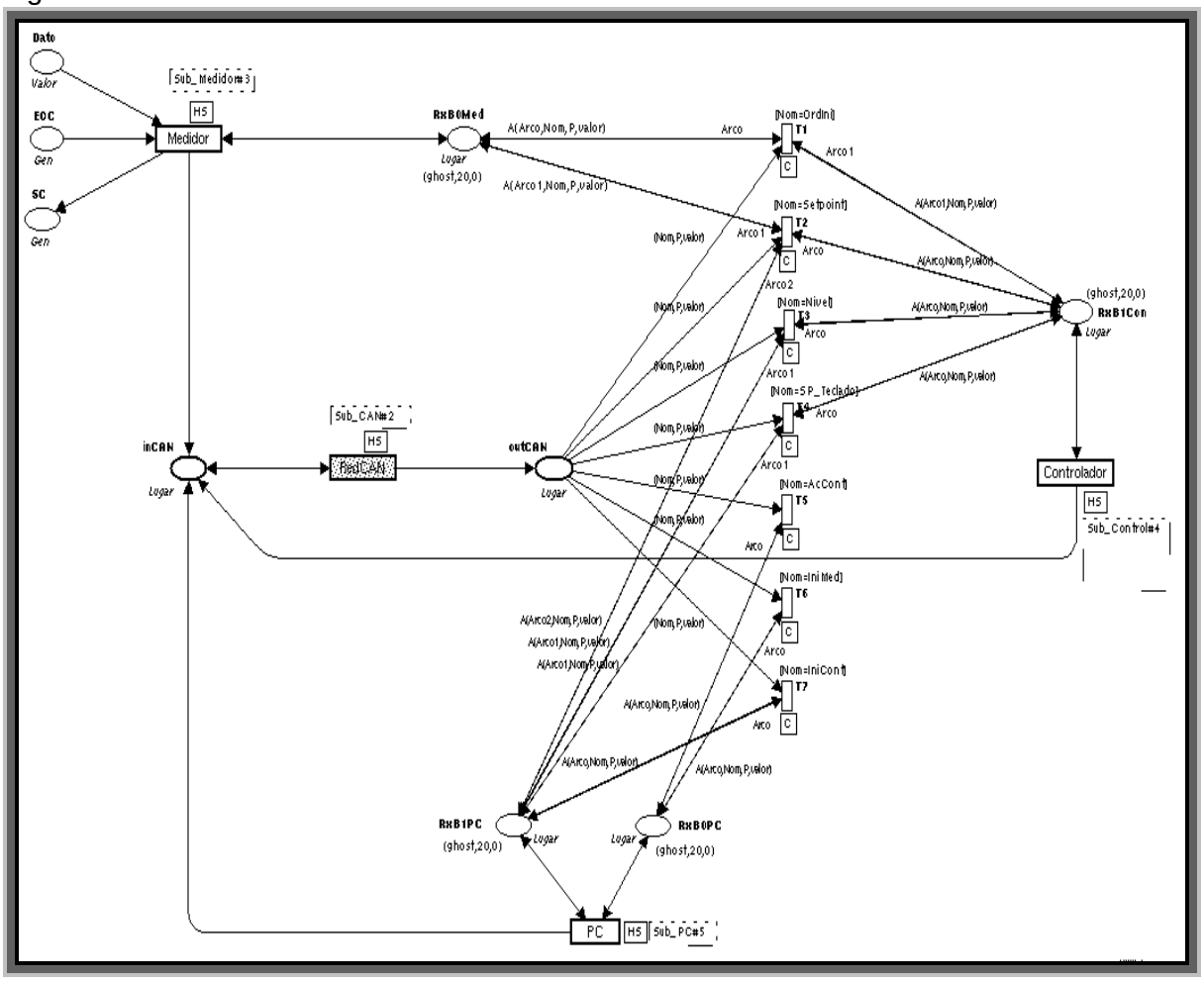
Figura 62. Página de Jerarquía del sistema modelado



De la página de jerarquía se puede deducir que el sistema está modelado en 24 páginas y que la declaración global se realizó en otra página diferente. Se define como página principal a *ControldeNivel* y a partir de ella se desprenden cuatro subpáginas: *sub_CAN*, *sub_Medidor*, *sub_Control* y *sub_PC*. Se observa que la única sub_pagina del segundo nivel que no tiene paginas jerárquicas internas es *sub_CAN*. En lo que sigue se dará una descripción del primer nivel de jerarquía y luego se analizarán los niveles jerárquicos más internos a partir del segundo nivel de jerarquía.

7.4.1 Pagina control de nivel (*ControldeNivel*) Esta es la página inicial de la red de Petri diseñada. En la figura 63 se puede observar el primer nivel del sistema, correspondiente a *controldenivel.*

Figura 63. Control de Nivel



La página *ControldeNivel* contiene cuatro paginas jerárquicas, representadas como las transiciones Medidor, RedCAN, PC y Controlador. También hay otros lugares que representan los puertos de entrada, salida ó entrada/salida de la subpágina. Se observan además las transiciones *T1*,*T2*,*T3*,*T4*,*T5*,*T6* y *T7*, las cuales permiten evaluar hacia que nodo va dirigido el mensaje que *RedCAN* envía y a la vez indica al nodo transmisor que el mensaje ha sido recibido correctamente por algún nodo.

La página sub_CAN contiene la implementación de la red CAN, y es tomada directamente de los componentes, sin realizarle ningún cambio. Por lo tanto este componente no será descrito en las siguientes secciones.

7.4.2 Pagina segundo nivel medidor (*Medidor*) Esta red se encarga de capturar los datos entregados por un sensor, visualizarlos internamente a través de un LCD, y de transmitirlos para que otros nodos (*Controlador* y *PC* a través de *RedCAN*) hagan uso de ese valor. Además implementa la posibilidad de cambio del nivel de referencia o setpoint

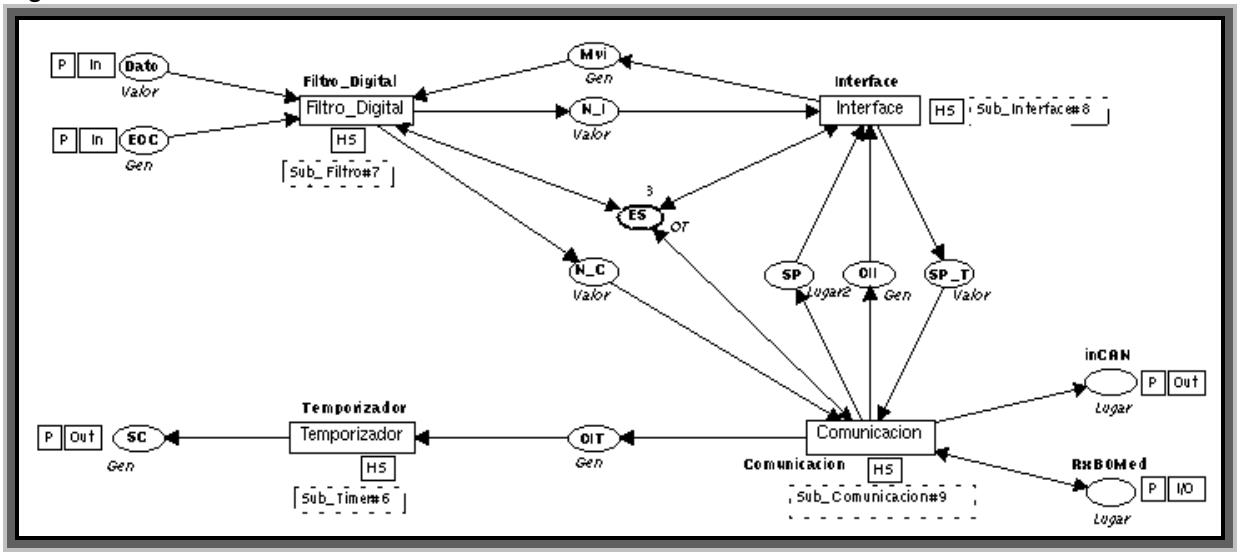
por medio de un teclado y un encoder. La red de Petri de este nivel se presenta en la figura 64.

Los mensajes que entrega este nivel hacia su exterior son:

-IniMed (Solicitud de Inicio del Medidor)
-Nivel
-SP_Teclado (Setpoint desde teclado)
-SC (Start Convertion)

Los tres primeros están dirigidos hacia la red CAN, y el último se dirige hacia un objeto externo que se encarga de leer físicamente el valor del sensor.

Figura 64. Medidor



Descripción de Lugares

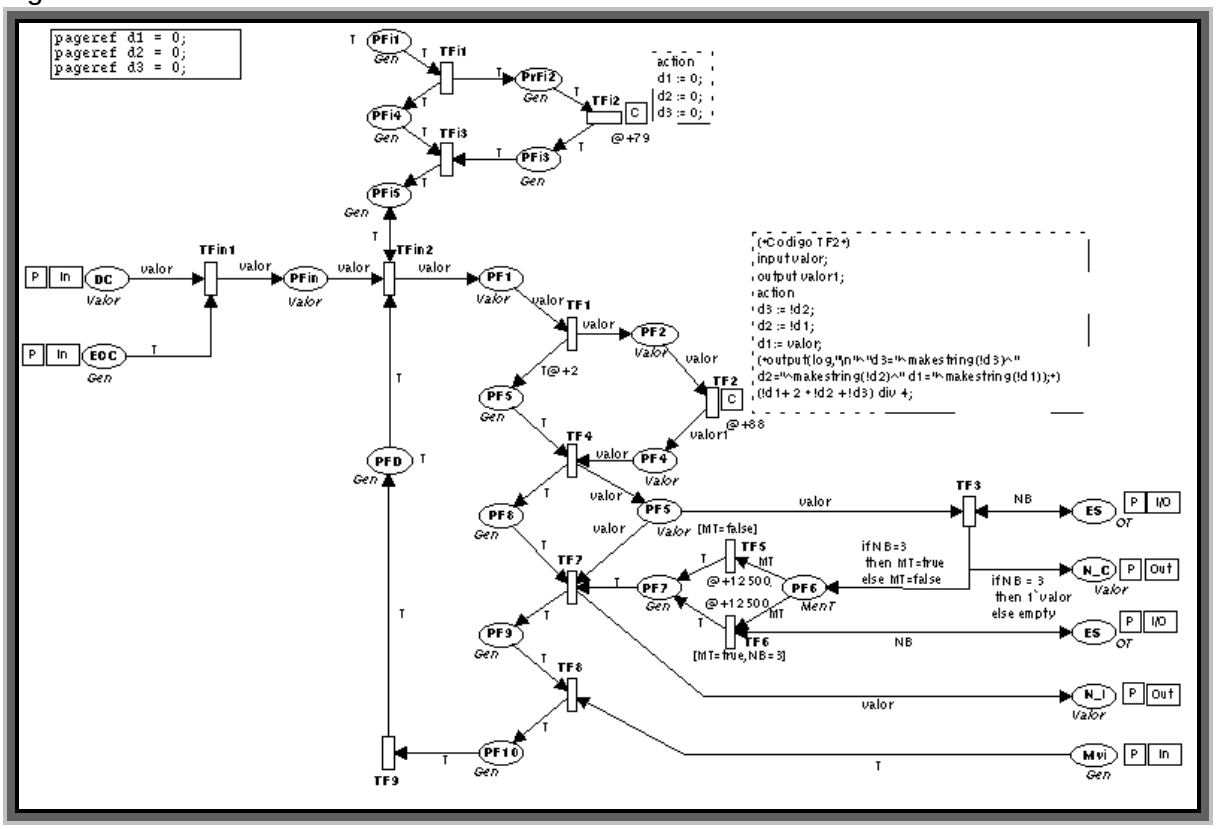
-Dato Dato disponible desde sensor o conversor
*EOC End Of Convertion
-N_I Nivel recibido y filtrado dirigido a interfase
-Mvi Mensaje Visualizado
-N_C Nivel recibido y filtrado dirigido a Comunicación
-ES Estado de la SPI
-SP SetPoint desde PC disponible
-OII Orden de Inicio a Interface Recibida
-SP_T Setpoint de Teclado interno disponible
-OIT Orden de Inicio a Teclado disponible

-RxB0 Buffer de Recepción Cero (0)

- Filtro de datos recibidos (*Filtro*) Este objeto recibe el valor digital entregado por el conversor o interfaz del sensor y le aplica una etapa de filtrado por medio de un filtro promediador (33). Luego de eso envía el dato a la interfaz SPI (si está disponible) dirigido a *Comunicación*, para que sea transmitido a la red, y finalmente utiliza a *Interfaz* para visualizar a través de un LCD el valor recibido. La figura 65 muestra esta página.

Para la etapa de filtrado, utiliza un filtro implementado en Hardware. En la *Comunicación* se invoca una función, la cual indica si la interfaz está disponible por medio del lugar *ES*, si está disponible se transmite y *Filtro* se queda esperando hasta que el dato haya sido transmitido por completo al ChipCAN, en caso contrario, simplemente se descarta la transmisión, y el dato se pierde. Los retardos asociados a *TF5* y a *TF6* representan los tiempos que se tardan en realizarse las comunicaciones entre el procesador y el chip CAN.

Figura 65. Filtro

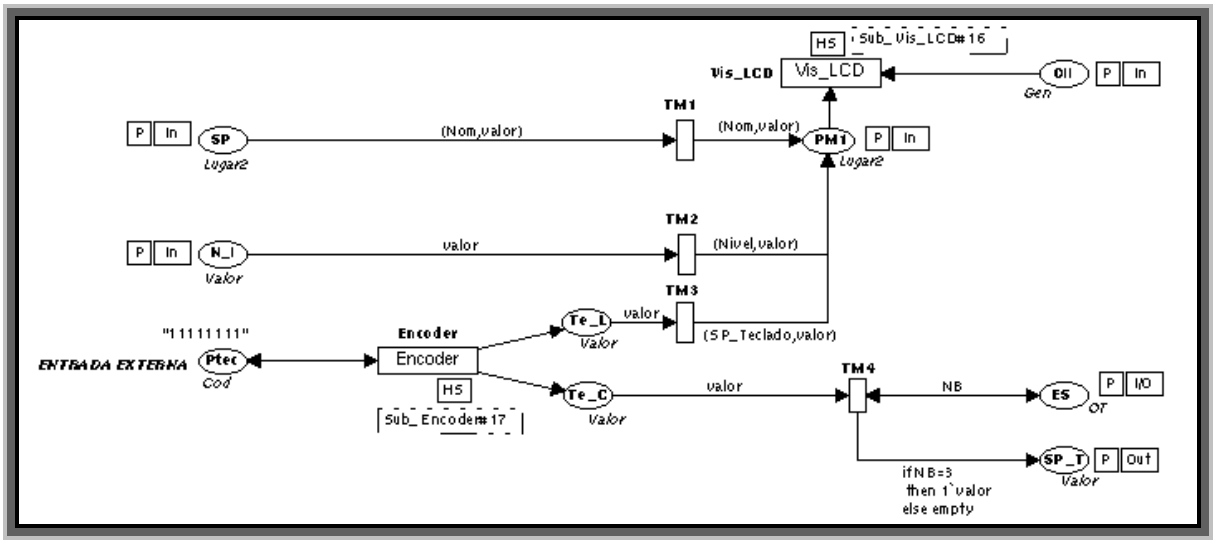


- Interface Se encarga de realizar la interface Operario-Sistema, a través de un LCD y un teclado. Los datos que recibe son: setpoint (desde *PC*), SP_Teclado (desde *Encoder*)

y Nivel (desde *Filtro*). Además recibe externamente los datos ingresados al teclado. La figura 66 representa este nivel.

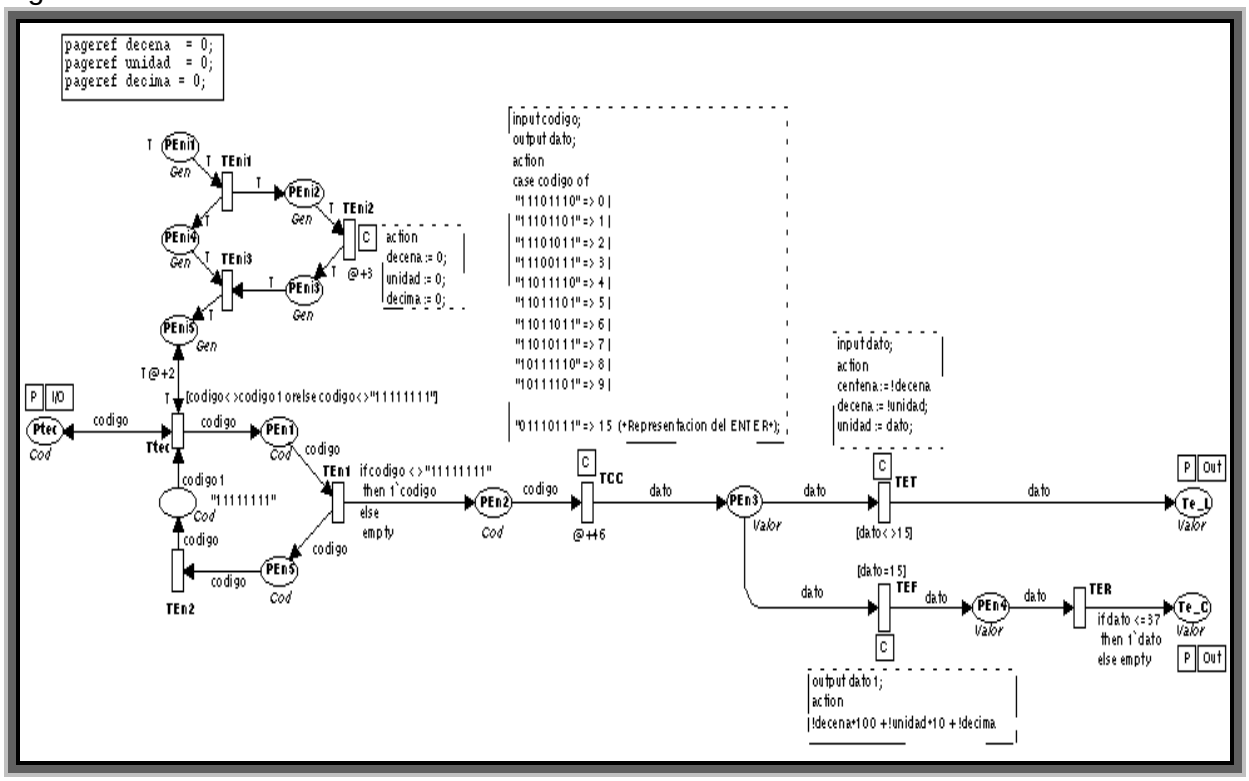
Para transmitir el dato a la red se debe realizar el mismo proceso de verificación con *ES*, para saber si pueden comunicarse con el chip CAN.

Figura 66. Interface



– Lector de teclado (*Encoder*) Este diagrama representa la forma como se realizó el proceso de lectura de la tecla que se presiona (el teclado es un componente físico externo). Se presenta en la figura 67.

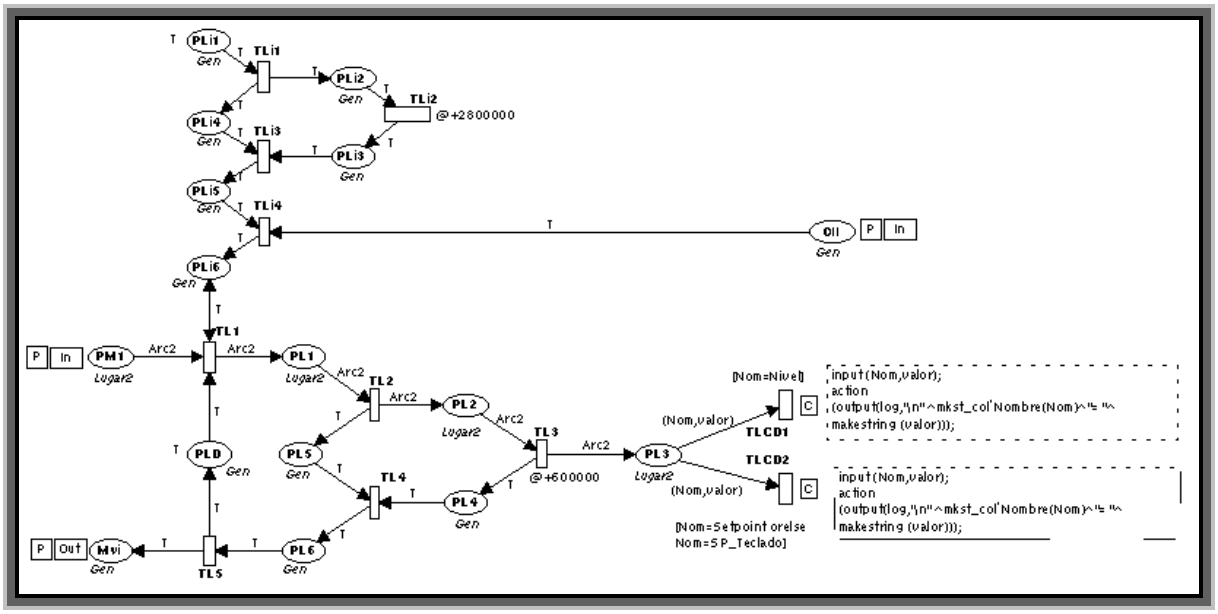
Figura 67. Encoder



Entre otras características, una de las más importantes es que si se trata de enviar un valor de nivel de referencia mayor al máximo nivel del tanque, este no se envía. Todas las teclas que se digitan se comunican al LCD, solo cuando se digita una tecla designada como “Enter”, se validan los tres últimos números (Decena, Unidad, Décima); el valor máximo para el tanque implementado físicamente es 37 cm.

– Visualizacion en LCD (*Vis_LCD*) Con efectos de visualización, se implementó una interfaz que permite observar algunos parámetros del sistema: nivel, Setpoint (desde PC) y SP_Teclado (teclado interno). La figura 68 ilustra esta red de Petri. Este objeto es sincronizado.

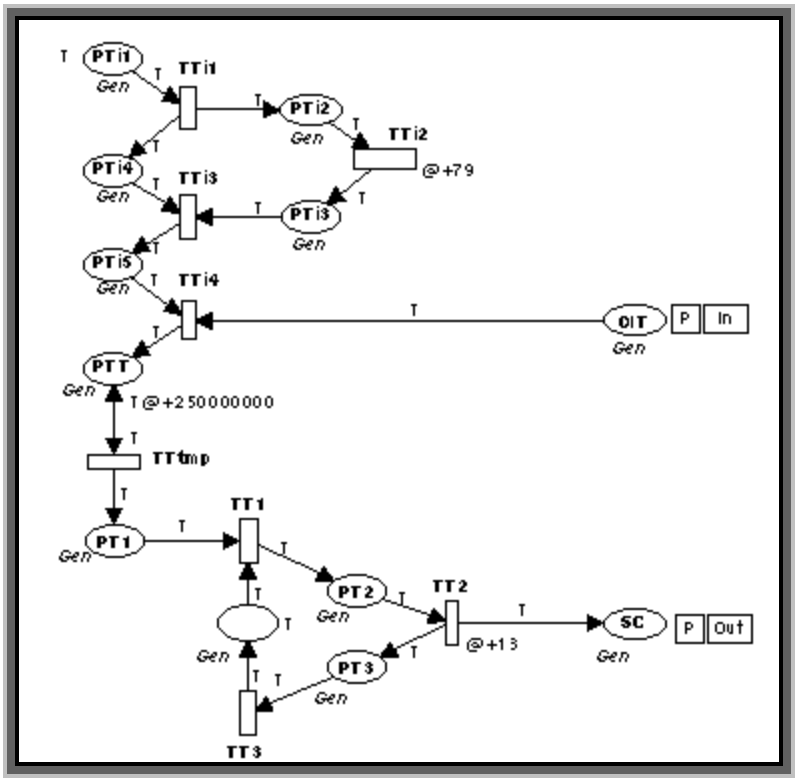
Figura 68. LCD



- Temporizador Este objeto se encarga de enviar periódicamente la orden de lectura a la interface que se comunica con el sensor (por ejemplo conversor). Espera que le llegue una orden de inicio para comenzar sus actividades. La red de Petri se puede observar en la figura 69.

La función periódica de este objeto determina el tiempo entre cada ciclo de muestreo (período de muestreo) sobre el que se va a trabajar dentro del sistema.

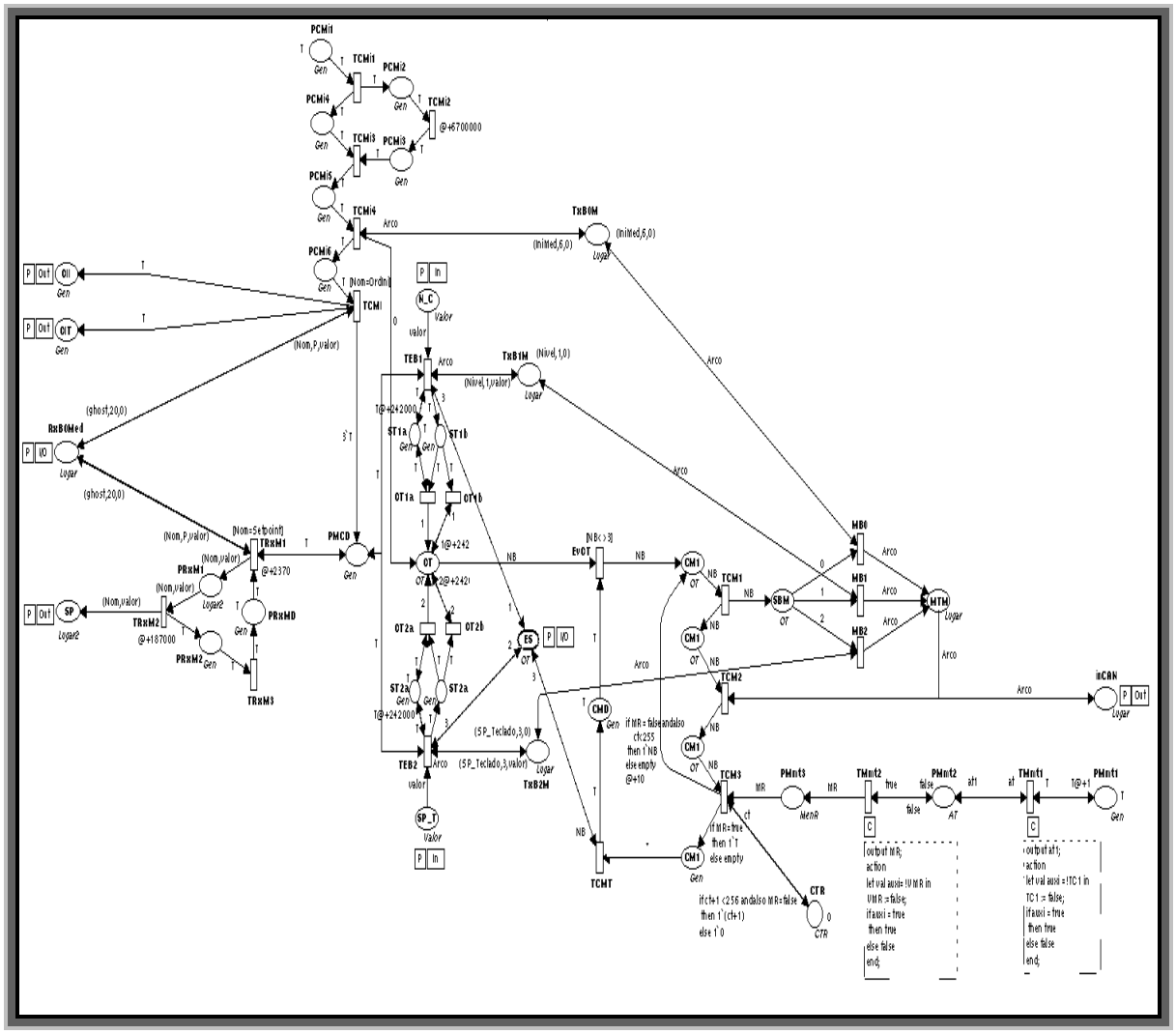
Figura 69. Temporizador



- Comunicación Este diagrama es el más complejo de todos, ya que implementa varios aspectos cercanos al protocolo CAN, a su configuración. Básicamente, se encarga de recibir un mensaje interno que se desea transmitir, y le adiciona algunos campos, para enviarlo a *RedCAN*, también permite tomar un mensaje de la red y entregar solo lo necesario al proceso interno que lo solicita. Su red de Petri se ilustra en la figura 70.

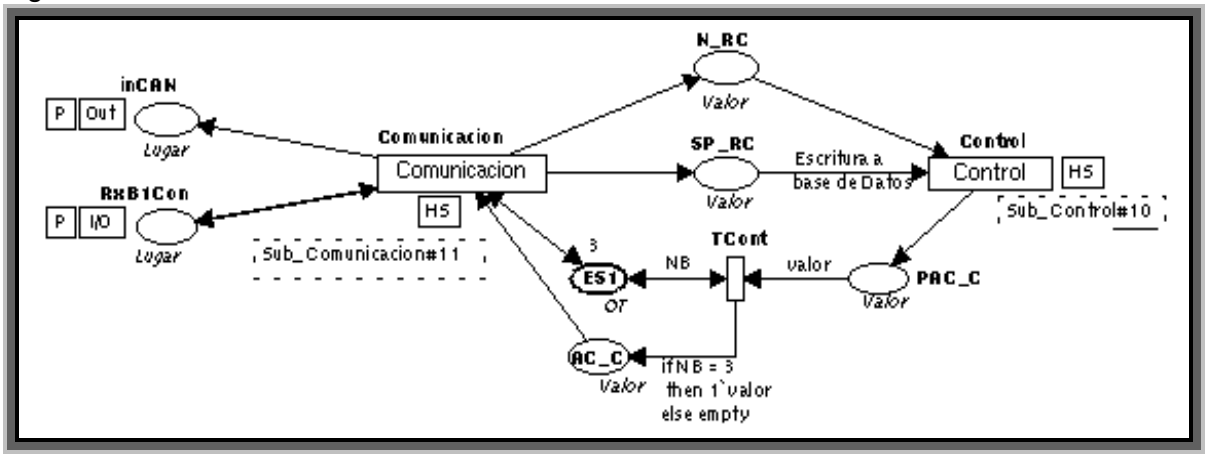
Todos los demás objetos comunicación funcionan de la misma forma, adicionando o quitando buffers de transmisión o de recepción según sea el caso.

Figura 70. Comunicación en medidor



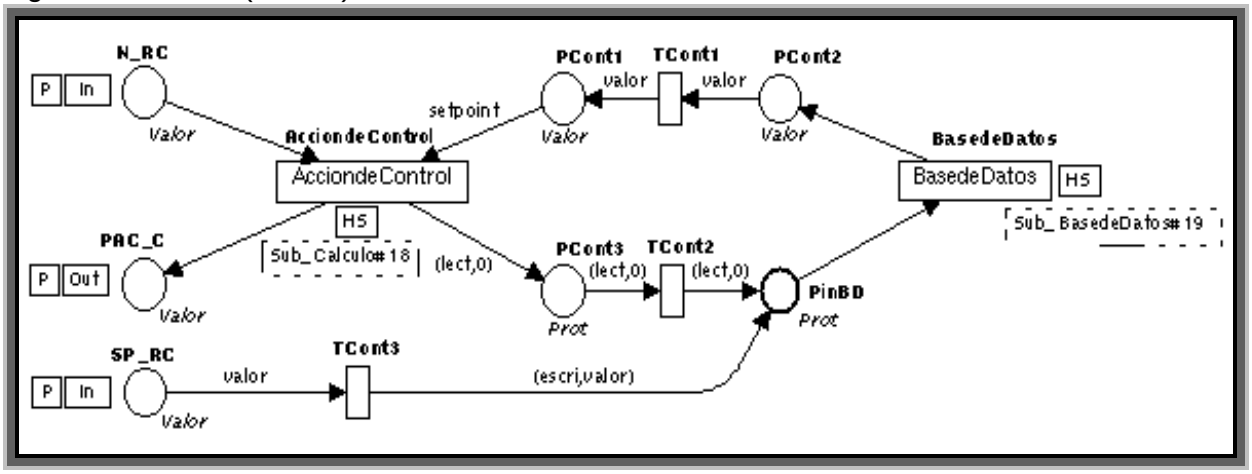
7.4.3 Pagina segundo nivel controlador (*Controlador*) Este pagina contiene básicamente dos sub_paginas: Comunicación y *Control*. Comunicación es más sencillo que el caso anterior, pero tiene la misma finalidad (no ser). La figura de la red de Petri de este nivel se presenta en la figura 71.

Figura 71. Controlador



- Control Recibe un valor de nivel, lo procesa, y determina la acción de control. Contiene una base de datos que permite proteger el valor de Setpoint (No se modelará debido a que no cambia con respecto a la presentada en los componentes componentes). La red de Petri modelada, se observa en la figura 72).

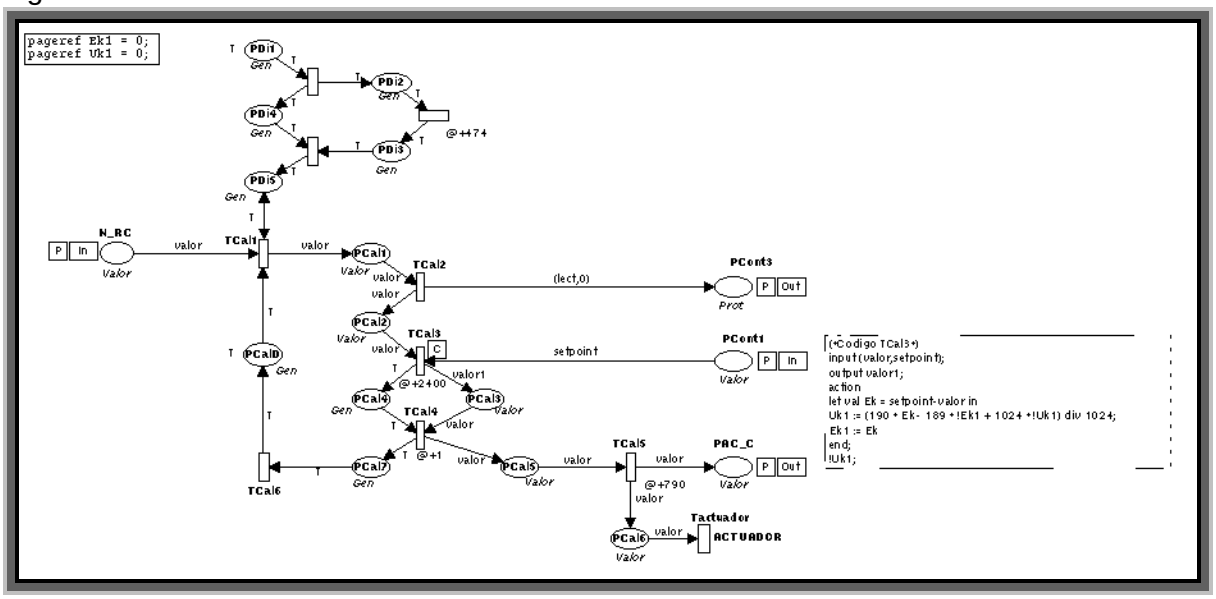
Figura 72. Control (interno)



– Acción de control Ejecuta una operación matemática para determinar el valor de la acción de control. Para este caso se utilizó un filtro promediador por sus variadas ventajas (33) Para este modelo, la red de Petri es la presentada en la figura 73..

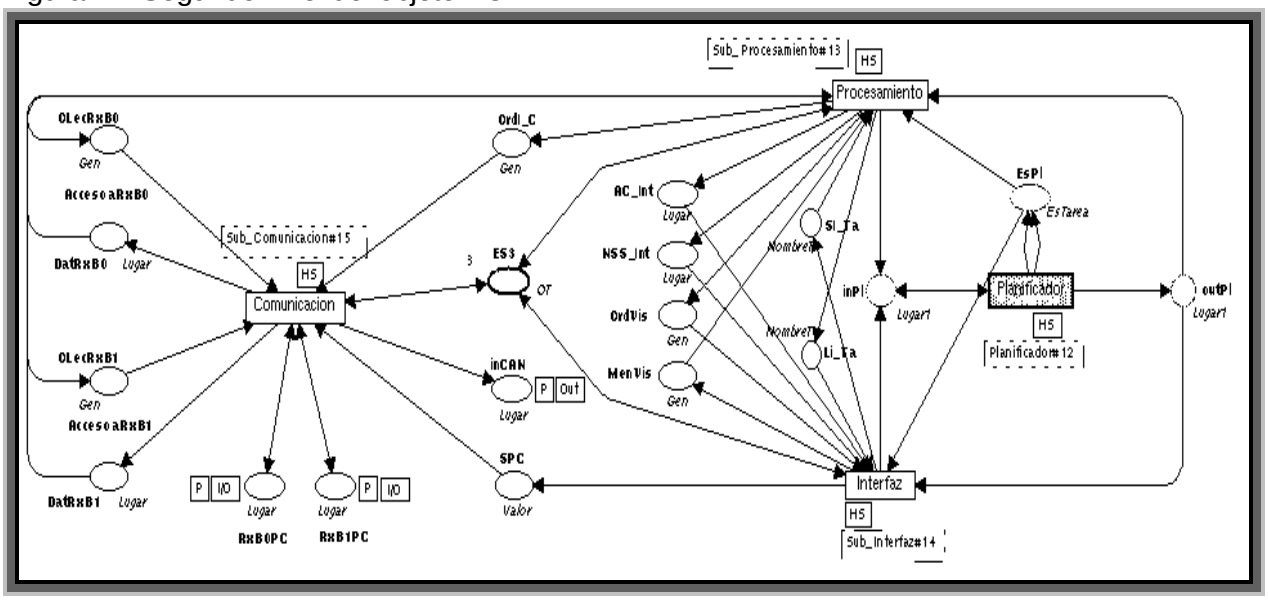
La acción de control calculada se envía hacia comunicación (si la interfaz SPI se encuentra disponibles).

Figura 73. Acción de Control



7.4.4 Pagina segundo nivel PC (*Controlador*) Para el objeto PC solo se visualizara el primer nivel, ya que faltan datos temporales para realizar completar el diseño. Para realizar la planificación se requiere de tiempo de computo de cada tarea, y sin él, no se puede realizar la planificación. La red de Petri se muestra en la figura 74.

Figura 74. Segundo nivel del objeto PC



Este nivel se compone de tres transiciones jerárquicas, cada una de las cuales permite realizar diferentes funciones como recepción de los datos, comunicar mensajes con la red y permitir observar la planificabilidad del sistema.

7.4.5 Declaración Para tener idea de la cantidad de funciones que se implementaron, a continuación se muestra, la declaración global de la red de Petri implementada.

```
(*DECLARACION GENERAL*)
color Nombre = with ghost|IniMed|IniCont|OrdIni|Setpoint|SP_Teclado|Nivel|AcCont; color Prioridad = int with ~1..20;
color Valor = int with ~50..50;
color Lugar = product Nombre*Prioridad*Valor timed;
color Gen = with T timed;
color Zero = bool timed;
color OT = int with 0..3 timed;
color MenR = bool;
color MenT = bool;
color Lugar3 = product Nombre*MenR timed;
color CTR = int with 0..256;
color AT = bool timed;

var Nom,Nom1: Nombre;
var P,P1,P2: Prioridad;
var valor,valor1 : Valor;
var Arco,Arco1,Arco2 : Lugar;
var cero,cero1 : Zero;
var NB,NB1 : OT; (*Numero de Buffer*)
var MR : MenR;
var MT : MenT;
var ct : CTR;
var at,at1 : AT;

val TC1 = ref false;
val TC2 = ref false;
val TC3 = ref false;
val VMR = ref false;

fun A(Ar,n,p,v)= if Ar=(ghost,20,0) then (n,p,v) else Ar;


(*CAN*)
val cter = ref 0;

(*MEDIDOR*)
color Lugar2 = product Nombre*Valor timed;

var Arc2 : Lugar2;
```

```
(*CONTROLADOR*)
color Valor1 = int with 0..1 timed;
color Tipo = with lect|escri;
color Prot = product Tipo*Valor timed;
color Cod = string timed;
var setpoint : Valor;
var T_Acce : Tipo;
var dato,dato1, dato2 : Valor;
var dato3 : Valor1;
var codigo, codigo1 : Cod;

(*PC*)
color ConTar = int;
color Barr = bool;
var x : ConTar;
var Sinc,Sinc2 : Barr;

(*Planificador*)
color NombreT = with IniIni|IniR_Datos|IniTecl|Sincronizacion|R_Datos|Teclado;
color TC = int with 0..100;
color TT = int with 0..100;
color TPrio = bool;
color Lugar1 = product NombreT*Prioridad*TPrio*TC*TT timed;
color Estado = unit with free timed;
color clk_tt = unit with t timed;
color ReIni = bool;
color OrdTarea = with inicio|expulsion;
color EsTarea = product NombreT*OrdTarea;
var NomT : NombreT;
var Pr : TPrio;
var tc : TC;
var tt,tt2 : TT;
var Arc1 : Lugar1;
var Sta : Estado;
var incr_tt : clk_tt;
var est : OrdTarea;

val cter1 = ref 0;
```

8. CONCLUSIONES

- Se resalta la importancia de tener una estrategia de trabajo cuando se realizan proyectos, con el fin de tener formas de evaluar las etapas que se desarrollan, y lograr por medio de análisis la predicción del comportamiento de dicho sistema; con el fin de corregir o mejorar los planteamientos iniciales.

- Los sistemas de control distribuido son una buena opción cuando se quieren reducir costos, aumentar seguridad, y otras mucha ventajas, sin embargo, requieren un cuidado especial debido a su variedad de componentes, es necesario buscar esquemas de modelamiento que permitan modelar sistemas de manera genérica, para de esta manera realizar un modelo global, donde se pueda analizar el comportamiento del sistema en conjunto.

- Las redes de comunicaciones y los planificadores de tareas influyen de manera significativa en el desempeño del sistema, por tanto, se requiere especificar dichos componentes y sus características al momento de realizar un diseño.

- Los sistemas en tiempo real requieren ser implementados con componentes que tengan un comportamiento predecible, y no solamente sistemas veloces.

- Las redes de Petri son una buena alternativa en el modelamiento de muchas clases de sistemas, para observar aspectos como los temporales y los funcionales. Pueden ser soportadas en HRT-HOOD, para facilitar el diseño.

- El planteamiento de algunos componentes específicos para el diseño de los sistemas tratados en esta tesis, permite al diseñador acercarse y reducir la complejidad que el diseño arquitectural le puede traer.

BIBLIOGRAFIA

BURNS, A y WELLINGS A.J. HRT-HOOD: A Structured Design Method for Hard Real-time Systems. Real-time and Distributed. Grupo de investigación en sistemas. Departamento de Ciencias de computación. Heslington, York : University of York, [sf]. 248 p.

BURNS, Alan. HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems. 1995. 44 p.

CHAPARRO, David. Redes de Petri (Un editor Gráfico). Monterrey, 1993, 159 p. Tesis (Maestría en Ciencias). Instituto Tecnológico y de Estudios Superiores de Monterrey.

Embarcadero Technologies, inc [En línea]. [San Francisco, U.S.A], 2003 [Citada en Abril de 2003]. Disponible en internet <http://www.embarcadero.com/support/what_is_uml.asp>

G. Rozenberg. Advances in Petri nets. New York, 1990.

GÓMEZ, Roberto. Introducción a los Sistemas Distribuidos. Monterrey, México : Instituto Tecnológico y de Estudios Superiores de Monterrey. Departamento de Ciencias Computacionales. 1996. 132 p.

GONZÁLES, Apolinar. Especificación de componentes mediante redes de Petri para el diseño y validación de sistemas de control de tiempo real. Valencia, 1999, 183 p. Tesis (Doctoral en ciencias de computación). Universidad Politécnica de Valencia. Departamento de informática de sistemas y computadores.

JENSEN, Kurt. An Introduction to the Theoretical Aspects of Coloured Petri Nets. Dinamarca, 1991. 56 p.

LIZCANO, Jorge y MACIAS, Hugo. Implementación de un modulo de sensado y de actuación para el control de nivel de un tanque mediante la técnica de codiseño Hardware – Software. Cali, 2003, 122 p. Tesis (Ingenieros eléctricos y electrónicos) Corporación Universitaria Autónoma de Occidente. Unidad académica eléctrica y electrónica.

MARTÍNEZ, Diego. Notas de Clase (Microcontroladores II). Cali : Corporación Universitaria Autónoma de Occidente, 2002.

_____ Planteamiento de un estilo de diseño formal para el desarrollo de Sistemas de Control Distribuidos. En : SEMINARIO INTERNACIONAL AVANCES Y TENDENCIAS EN ELECTRÓNICA ALTO NIVEL. Corporación Universitaria Autónoma de Occidente. Santiago de Cali. Agosto 14 a 16 de 2000.

Object Management Group, inc (OMG) [En línea]. Marzo de 2003 [Citada en Abril de 2003]. Disponible en internet <http://www.omg.org/uml/>.

SALINAS, patricio. Tutorial UML [En línea]. [Santiago de Chile, Chile], octubre de 1996 [Citada en noviembre de 2002]. Disponible en internet. <http://www.dcc.uchile.cl/~psalinas/uml/ >.

TERES, Luis; TORROJA, Yago; LOCOS, Serafín y VILLAR, Eugenio. Lenguaje estándar de Diseño Electrónico VHDL. España : McGraw-Hill, 1998. 239 p.

TÖRNGREN, Martin; ELKHOURY, Jad y SANFRIDSON, Martin y REDELL, Ola. Modelling and Simulation of Embedded Computer Control Systems : Problem Formulation. Mechatronics Lab. Estocolmo, Suecia. 128 p;

Universidad de Aarhus [En línea]. [Dinamarca], marzo de 2003 [Citada en febrero 2002]. Disponible en internet <http://www.daimi.au.dk/Cpnets>

ANEXOS

ANEXO A
EJEMPLO DE DIAGRAMA DE USOS DE CASOS

Como ejemplo de un modelo de los diagramas de Uso de Casos se presenta en este apéndice, el caso de una Máquina Recicladora:

El sistema se encarga del manejo de una máquina de reciclamiento de botellas, tarros y jabas. Se debe controlar y/o aceptar:

Registrar el número de ítems ingresados.

Imprimir un recibo cuando el usuario lo solicita:

-Describe lo depositado
-El valor de cada ítem
-Total

El usuario/cliente presiona el botón de comienzo

Existe un operador que desea saber lo siguiente:

-Cuantos ítemes han sido retornados en el día.
-Al final de cada día el operador solicita un resumen de todo lo depositado en el día.

El operador debe además poder cambiar:

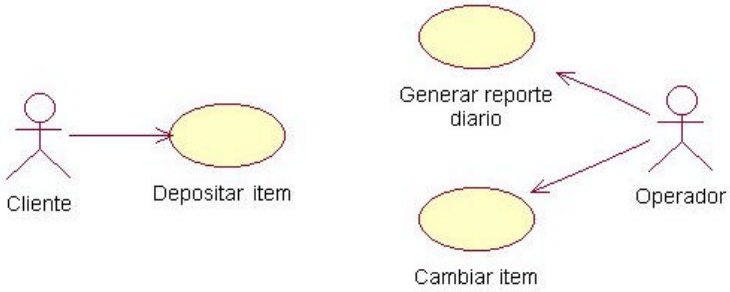
-Información asociada a ítemes.
-Dar una alarma en el caso de que:
-Item se atora.
-No hay más papel.

Como una primera aproximación, en la figura 75 se identifican a los actores que interactúan con el sistema.

Figura 75. Ejemplo (primer análisis)

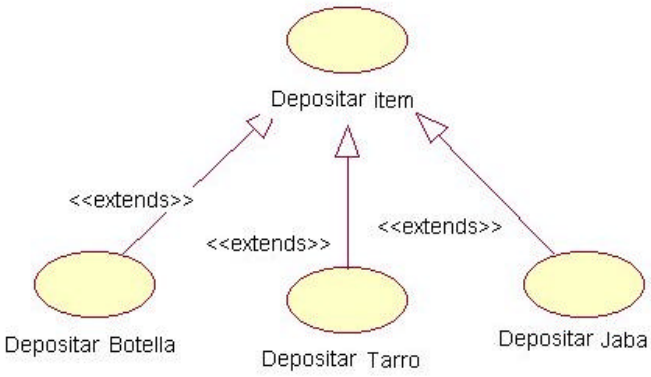
Luego, un Cliente puede Depositar Items y un Operador puede cambiar la información de un Item o bien puede Imprimir un informe (Figura 76).

Figura 76. Ejemplo (segundo análisis)



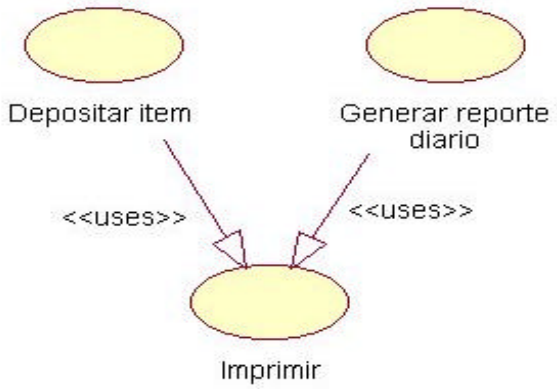
Además se puede observar que un item puede ser una Botella, un Tarro o una Jaba (Figura 77).

Figura 77. Ejemplo (tercer análisis)



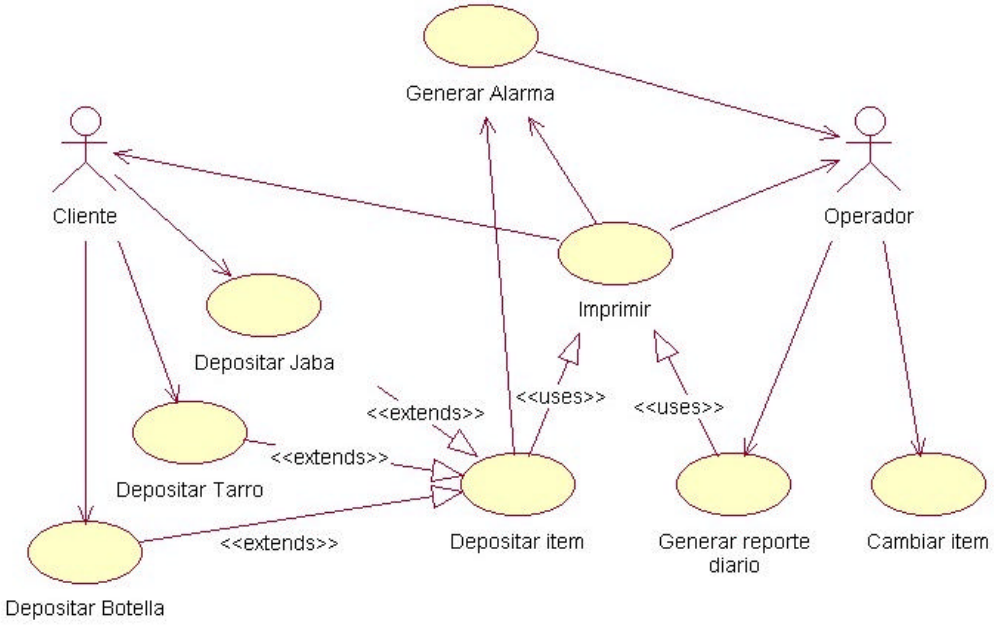
Otro aspecto es la impresión de comprobantes, que puede ser realizada después de depositar algún item por un cliente o bien puede ser realizada a petición de un operador (figura 78).

Figura 78. Ejemplo (cuarto análisis)



El diseño completo del diagrama casos de uso se presenta en la figura 79.

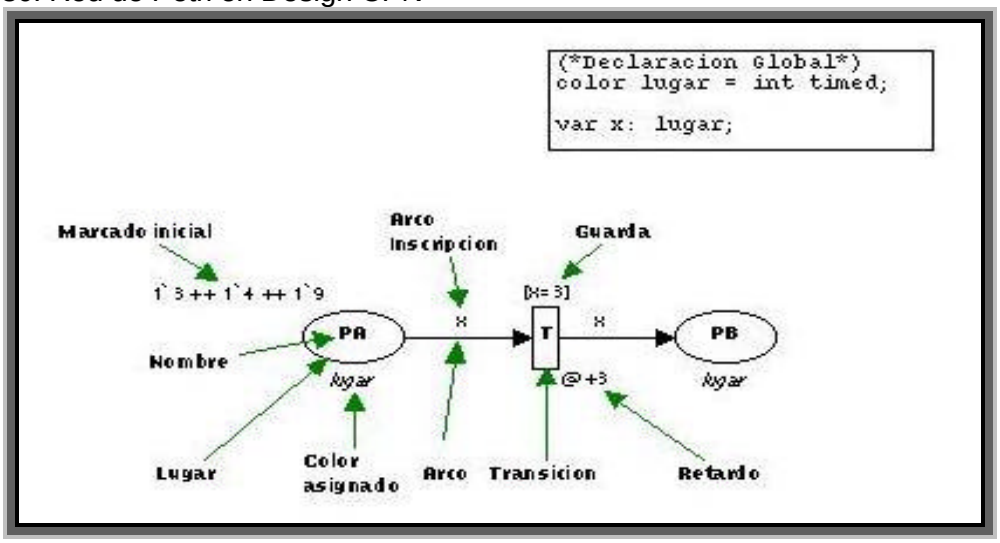
Figura 79. Ejemplo (diseño final)



ANEXO B
INTERPRETACIÓN DE LAS REDES DE PETRI EN DESIGN/CPN

Una red de Petri Coloreada está compuesta de: lugares, transiciones, arcos, tokens, regiones de declaración (global, local, temporary), guardas, expresiones de arco, páginas de jerarquía.

Figura 80. Red de Petri en Design CPN



Las *regiones de declaraciones* son las secciones donde se realiza la declaración de colores, variables, valores, funciones, etc.

Las *expresiones* de arco son términos que definen el peso del arco, lo condicionan, definen que tipo de tokens pueden utilizarlo, etc.

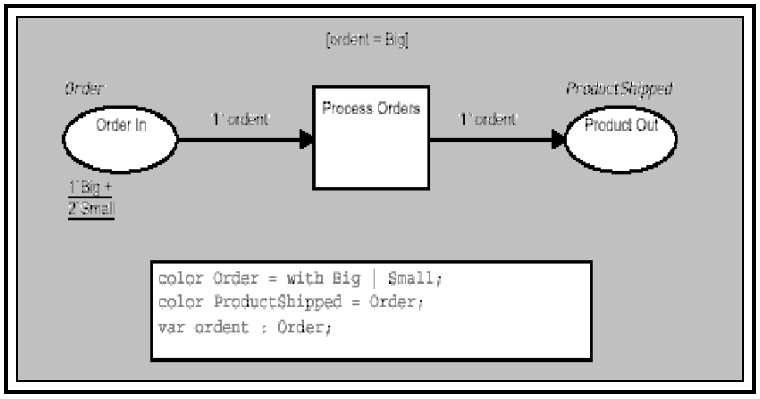
Los *guardas* son expresiones que evalúan condiciones de activación de las transiciones. Por ejemplo, una transición que tenga de entrada a un lugar que está declarado con el color *int* (entero), y están relacionados con un arco que en su arco inscripción tiene la variable x, entonces sería posible evaluar condiciones como: [x<3], [x=9], [x<=2], [x>0,x<8] (se deben cumplir las dos condiciones), [x=4 orelse x=8], etc. Si a condición no se cumple, así haya tokens a la entrada de la transición, ésta no se activará.

Las páginas de jerarquía definen las relaciones en estructuras jerárquicas de redes de Petri. En estas se puede observar la relación de parentesco entre los diferentes niveles realizados. Es muy útil como página guía del modelo.

Ejemplo

En la figura 81 se presenta una red de Petri sencilla.

Figura 81. Ejemplo de Red de Petri Coloreada



En la zona de declaraciones, se definen dos colores (*Order*, *ProductShipped*) y una variable (*ordent*). El color asociado al lugar de nombre Order In (izquierdo en la figura) es *Order*, y el segundo de nombre Product Out tiene un color *ProductShipped*. También se puede observar que las inscripciones de arco utilizan a la variable *ordent*, para dejar pasar tokens *Big* y *Small*. Finalmente, la transición tiene un guarda (*[ordent=Big]*) que permite su activación solo por tokens *Big*.

ANEXO C
USO DE DESIGN/CPN (GUÍA PRÁCTICA)

En este anexo, se tratarán los pasos básicos para crear y simular una red de Petri Coloreada por medio del software *Design/CPN*. Las recomendaciones aquí expuestas, corresponden a mi propia experiencia con el programa.

Para descargar el software, y tener una profundización en los diferentes aspectos de sus manejo, se recomienda visitar la página web de la universidad de Aarhus (Dinamarca).

INSTALACIÓN

Design/CPN puede ser instalado sobre diferentes sistemas operativos, entre los que se encuentran entre otros Solaris y Linux. En adelante se tratará solo la utilización de Linux, ya que fue el software utilizado para esta tesis. La versión 4.0.5 de Design/CPN permite la instalación sobre Linux RedHat, SuSE (utilizado en este trabajo), Caldera, Storm, los cuales son mencionados por haber sido probados.

La instalación (en Linux) se puede lograr utilizando el auto-instalador de formato *RPM* (DesignCPN-4[1].0.5-1.i386.rpm, para la versión 4.0.5 del software), o por medio del archivo comprimido *TAR* (DesignCPN-4[1].0.5-linux.tar, para la versión 4.0.5 del software). Una recomendación es descargar el software directamente de la página de la universidad de Aarhus, o utilizar una versión reciente (superior a la 4.0).

Cuando se utiliza el *TAR*, no se debe olvidar (después de su descompresión) modificar el archivo “cpn” y fijar la ruta de la carpeta que contiene el archivo descomprimido. Una explicación más detallada de la instalación por este método (de descompresión) se encuentra en un archivo de instalación que aparece en la descompresión.

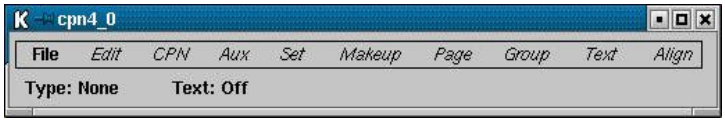
PRINCIPIOS DE USO DE DESIGN/CPN

Para iniciar la utilización de Design/CPN, se puede utilizar la consola de Linux, o ir directamente hacia la ubicación de los archivos de instalación y dar *click* sobre “cpn”.

Para utilizar la consola, se debe dar la ruta donde se encuentran ubicados los archivos de Design/CPN, y luego ejecutar “cpn”: *<Ruta de Ubicación> ./cpn*.

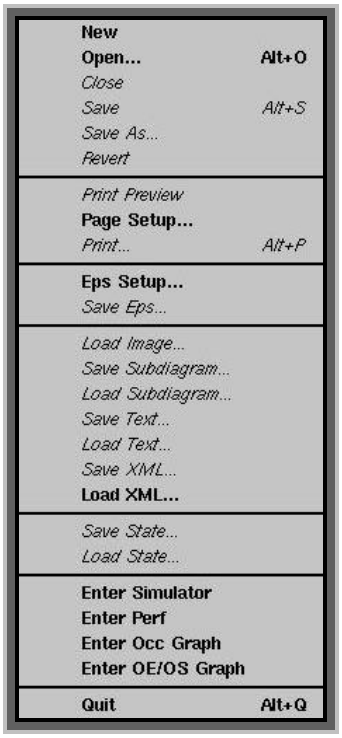
Una vez el software es inicializado, aparece una ventana como la ilustrada en la figura 82.

Figura 82. Ventana de Design/CPN



Para crear una nueva red de Petri Coloreada se utiliza el menú *File* y la opción *New.*Si se desea abrir un archivo existente se utiliza *Open* del mismo menú. Figura 83.

Figura 83. Opciones del menú *File*



Luego de tener un diagrama abierto, se pueden insertar los lugares, transiciones y arcos necesarios para la red, por medio del menú *CPN*. Figura 84.

Figura 84. Opciones del menú *CPN*

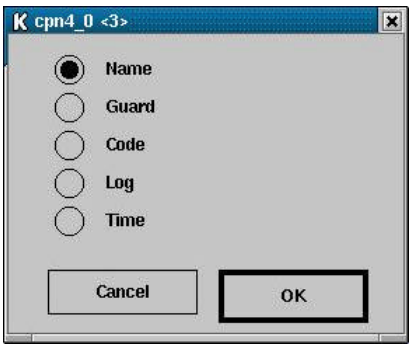
Para modificar o adicionar las propiedades de cada nodo (Transición, lugar o arco), se debe señalar el nodo y dentro del menú *CPN*, utilizar la opción *CPN Region*, o simplemente señalar el nodo y presionar ALT+R.

Las opciones para un lugar son: nombre, color del lugar, marcas iniciales; en la figura 85 se pueden observar las diferentes opciones. El nombre no es obligatorio, el color asignado al lugar restringe la recepción de tokens a los de su tipo (color) y es obligatorio, y la marca inicial se coloca dependiendo de las condiciones iniciales del modelo.

Figura 85. Regiones de un lugar

Para una transición las opciones se muestran en la figura 86. El nombre es opcional, los guardas condicionan la ocurrencia de la transición, el código se puede utilizar para realizar modificaciones en los tokens de salida de la transición o para modificar variables referenciadas (entre otros fines), la región de registro (log) permite visualizar todo lo que desde el código se indique, el tiempo corresponde al retardo que se le va a adicionar a los tokens temporizados.

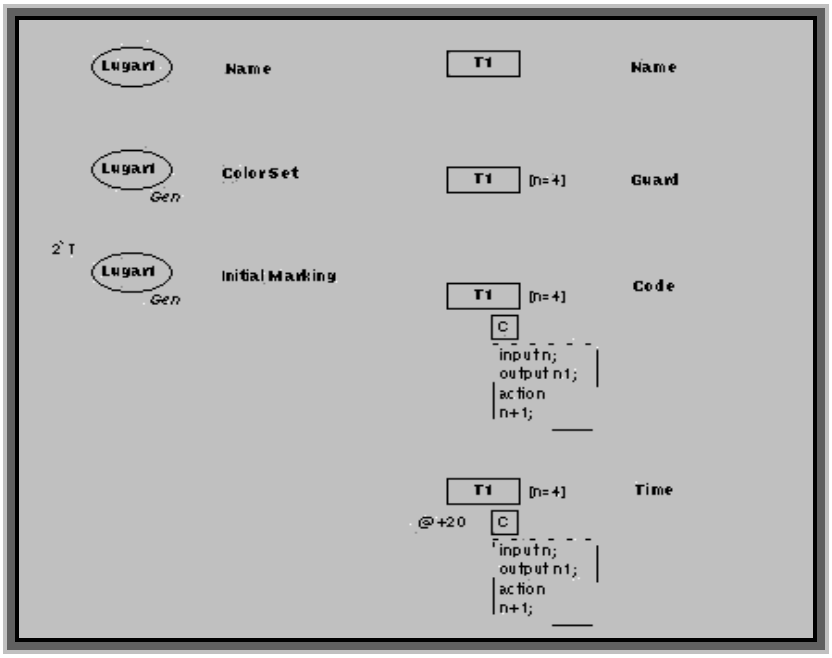
Figura 86. Región de una transición



Los arcos permiten solo asignar expresiones de arco, para indicar su peso, para realizar condiciones, para incluir retardos, etc.

En la figura 87 se puede observar la asignación de diferentes regiones a un lugar y a una transición.

Figura 87. Ejemplos de asignaciones de regiones a un lugar y a una transición

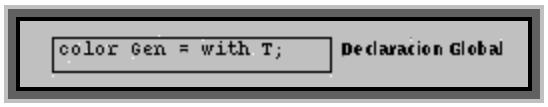


Para realizar las declaraciones bien sea locales, temporales o globales, se utiliza la opción *Declaration Node* del menú *CPN* (Figura 88).

Figura 88. Opciones de declaraciones

La declaración *Global* es donde se describen los colores, variables, funciones y valores que van a ser utilizados dentro de la red de Petri. Es obligatoria y cada diagrama solo puede contener una declaración de este tipo. Un ejemplo de este tipo de declaración es el de la figura 89.

Figura 89. Ejemplo de declaración *Global*



Cada diagrama puede tener una declaración *Temporary*. Se diferencia de la declaración global en que ésta se puede utilizar en medio de una simulación (utilizando el editor, después de haber estado simulando), y no provoca una nueva revisión de sintaxis de todo el diagrama. Si la declaración global es modificada durante una simulación, el diagrama será revisado en su totalidad de nuevo, así no se haya modificado nada del diagrama. En otras palabras, la declaración Temporary es "temporal"; finalmente después de realizar las pruebas necesarias, es aconsejable incluir el contenido de esta declaración en la declaración global.

La declaración *Local* se utiliza para declarar variables locales. Dichas variables solo pueden utilizarse en segmentos de código. La sintaxis de declaración es la siguiente:

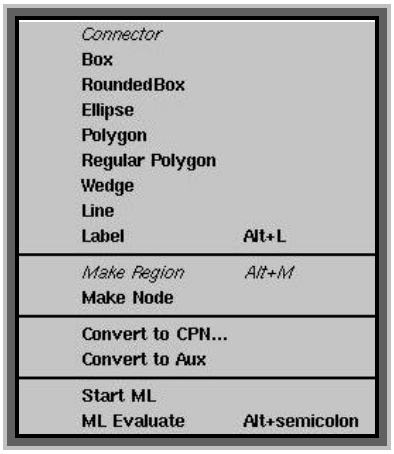
pageref Id = exp; (Las variables de este tipo son las mismas para todas las instancias que las mencionan).

instref Id = exp; (Las variables de este tipo son diferentes para todas las instancias que las mencionan).

Donde *exp* indica el tipo de variable que se define; por ejemplo, si exp es 0 (o cualquier número entero), la variable será de tipo entero, si es 0.4 (o cualquier número real) será real, etc. Cada pagina puede tener una declaración local, la cual es totalmente independiente de las declaraciones locales de otras paginas (Se pueden utilizar los mismos nombres para las variables en las diferentes declaraciones locales del diagrama).

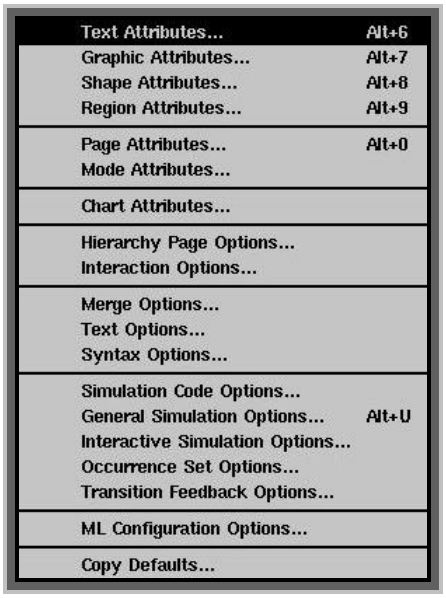
Otros menús que son útiles en la implementación de la red de Petri son los siguientes.

Figura 90. Menú *Aux*



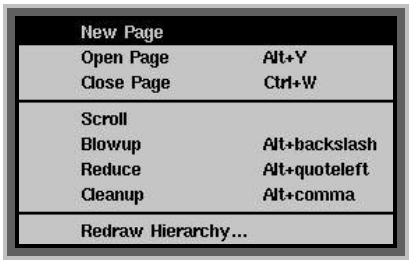
El menú *Aux* (figura 90) menú permite realizar diferentes figuras o escribir comentarios sin que estos sean tomados en cuenta por el simulador. Los objetos que se dibujen pueden ser convertidos luego en componentes de la red de Petri por medio de la opción *Convert to CPN*; también se puede realizar el proceso inverso: convertir componentes CPN a Auxiliar por medio de *Convert to Aux*.

Figura 91. Menú *Set*



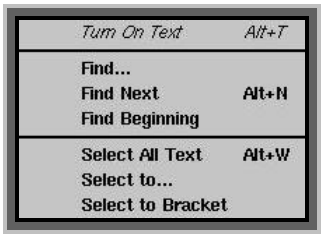
El menú *Set* (figura 91) permite realizar configuraciones de diferentes campos del sistema. Por ejemplo con la opción *Page Atributes*, se puede cambiar el nombre de la página donde se está graficando la red, se le pueden modificar sus dimensiones y se cambian diferentes aspectos relacionados con la pagina. Con la opción *Ocurrence Set Options* se puede configurar la manera en que la ocurrencia del diagrama se presentará, con todos los porcentajes del submenú de esta opción en 100% el sistema ejecutará la simulación de manera concurrente en un solo paso. Hay muchas opciones importantes en este menú.

Figura 92. Menú *Page*



El menú de la figura 92 permite abrir la página de jerarquía para un diagrama nuevo, por medio de la opción *Open Page*, permite ampliar o reducir la presentación de la pagina en observación, etc.

Figura 93. Menú *Text*



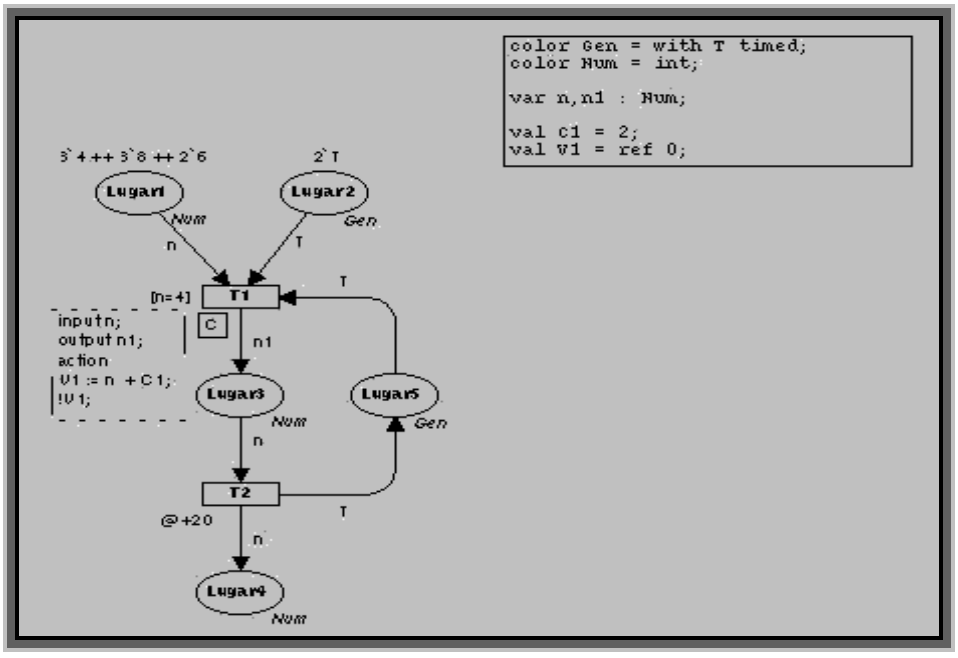
La opción *Turn On Text*, del menú *Text* permite realizar modificaciones en cualquier campo textual (regiones de transiciones y lugares, labels, declaraciones, etc.) de la red de Petri. Las otras opciones permiten realizar otras funciones elementales de texto.

A continuación se presenta un ejemplo donde se utilizan diferentes componentes y se dará una explicación de la forma en que se simulan las redes de Petri Coloreadas en Design/CPN.

EJEMPLO.

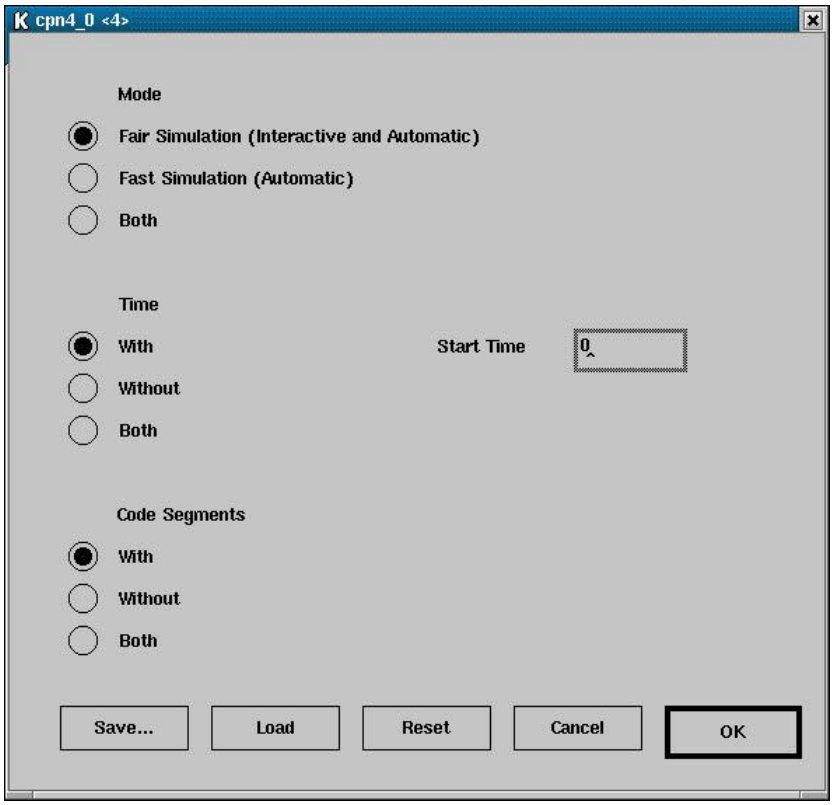
La figura 94 ilustra un diagrama completo realizado en el *editor* de Design/CPN, el cual será simulado posteriormente en el *simulador* del mismo programa.

Figura 94. Ejemplo: estructura de una Red de Petri Coloreada



En la declaración global se utiliza un colore temporizado, *Gen*, razón por la cual, antes de realizar un chequeo de sintaxis se debe configurar el programa para que permita la simulación. Para este fin se utiliza la opción *Simulation Code Options* del menú *Set*. El submenú ilustrado en la figura 95 corresponde a la opción mencionada.

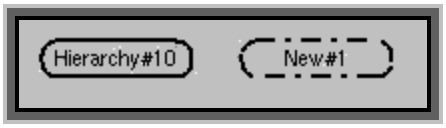
Figura 95. Opciones de simulación de código.



En el submenú se debe seleccionar la opción *With* de la sección *Time*, para que el editor interprete que la simulación incluye colores temporizados. Si esto no se hace, se presentará un error.

La página de jerarquía permite observar las diferentes páginas que la red de Petri diseñada incluye y su relación. La figura 96 ilustra la página de jerarquía de la red de Petri de este ejemplo.

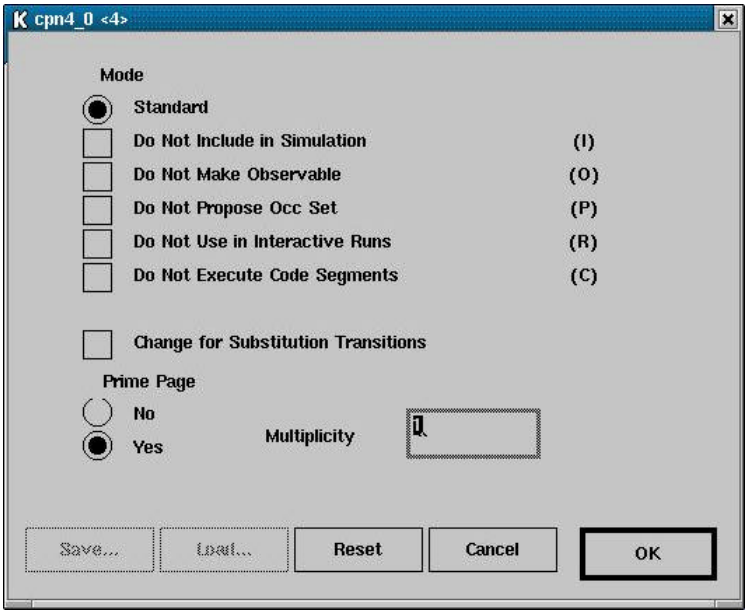
Figura 96. Ejemplo: contenido de la página de jerarquía



Hierarchy #10 corresponde a la página de jerarquía y *New#1* es la página donde se encuentra la red de Petri implementada.

Otro aspecto importante para realizar la simulación corresponde a la asignación de la página maestro del diagrama. Esto se realiza por medio de la opción *Mode Atributes* del menú *Set* (figura 97). Primero se selecciona la página que va a ser la página maestro (puede hacerse desde la página de Jerarquía) y luego se utiliza la opción mencionada. En el submenú se selecciona *Yes* en la sección *Prime Page*.

Figura 97. Submenú *Mode Atributes*



En el submenú de la figura 97, se puede también seleccionar opciones de página que permiten restringir el uso de la página durante la simulación.

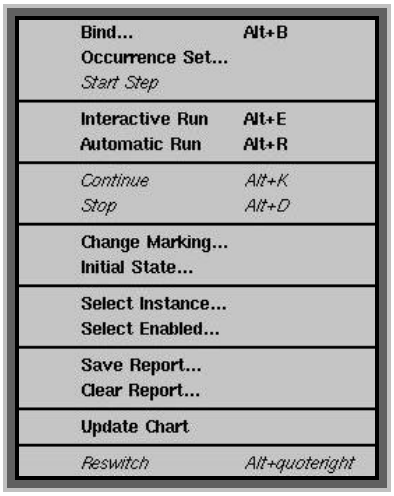
Después de seleccionar la opción de página principal o maestra, la página de jerarquía luce como en la figura 98.

Figura 98. Ejemplo: contenido de la página de jerarquía con página maestra

La revisión de sintaxis se puede realizar sin la asignación de página maestra. Esta revisión se realiza por medio de *Sintax Check* del menú *CPN*. Si se desea realizar una simulación, basta con utilizar *Enter Simulator*, del menú *File*, y la sintaxis será chequeada de paso. Si durante la revisión se presenta algún error, en la página de jerarquía se indicará la página que lo contiene.

Una vez se ha ingresado al simulador aparece un nuevo menú: *Sim*. Este menú se ilustra en la figura 99. Con este menú se realiza la ejecución de la simulación, se puede inicializar el diagrama en cualquier instante, o se pueden realizar otras funciones relacionadas con la simulación.

Figura 99. Menú *Sim*



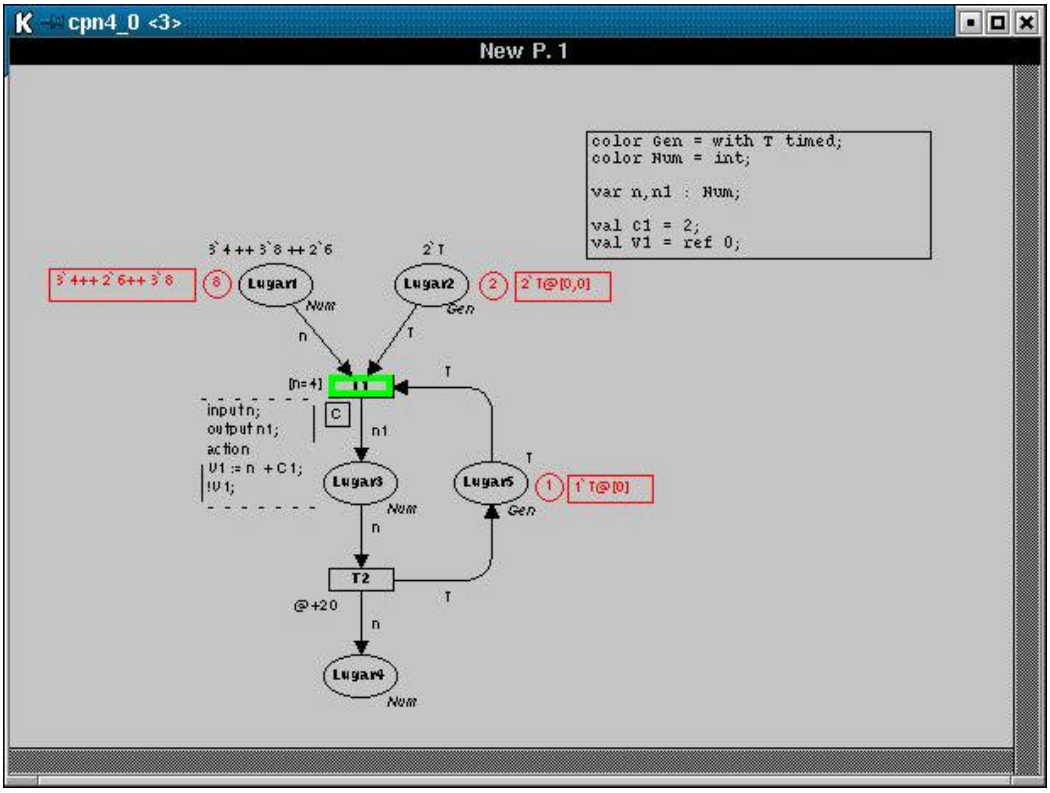
La ejecución manual se puede realizar señalando una transición y utilizando la opción *Bind* del menú *SIM*, o por medio de *Ocurrence Set* del mismo menú. En la primera opción solo se disparará la transición señalada, mientras en la segunda se pueden realizar varios disparos al tiempo y en el mismo paso (si así se encuentra configurado el simulador).

También se pueden realizar simulaciones interactivas o simulaciones automáticas. Las primeras permiten observar la ejecución de la red de manera dinámica, mientras las segundas solo muestran el resultado después de que se ha llegado a un estado de

bloqueo (cuando no hay más transiciones habilitadas) o cuando se ha cumplido alguno de los criterios de detención de la simulación (configurables en la opción *General Simulation Options* del menú *Set*). Los criterios de detención también se aplican en la simulación interactiva. Para la simulación interactiva se pueden realizar los ajustes requeridos, para que la simulación se detenga al inicio, durante o al final del disparo de cada transición; también se puede configurar para que la simulación no se detenga. Esta configuración se realiza en el menú *Interactive Simulations Options*.

La figura 100 ilustra la red de Petri cuando se ha ingresado al simulador.

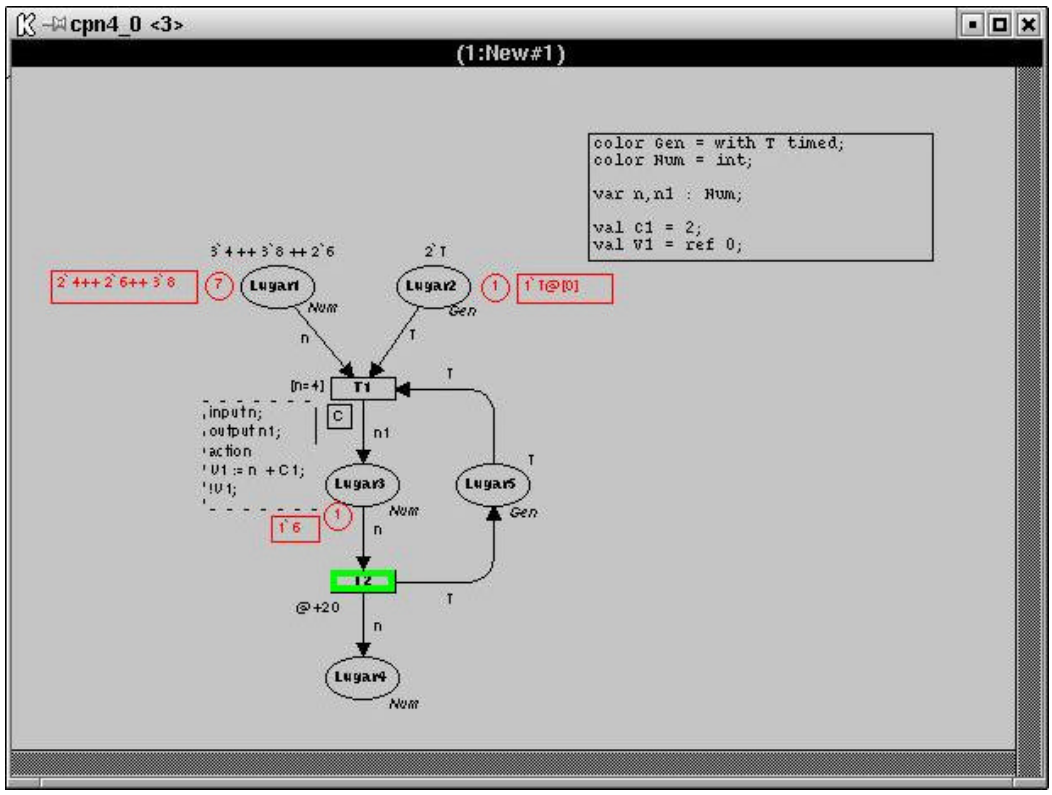
Figura 100. Ejemplo: Red de Petri en el simulador



En la figura 100 se observa la cantidad de tokens junto al lugar correspondiente encerrados en un circulo y su descripción en un rectángulo. Las transiciones activas se encuentran resaltadas.

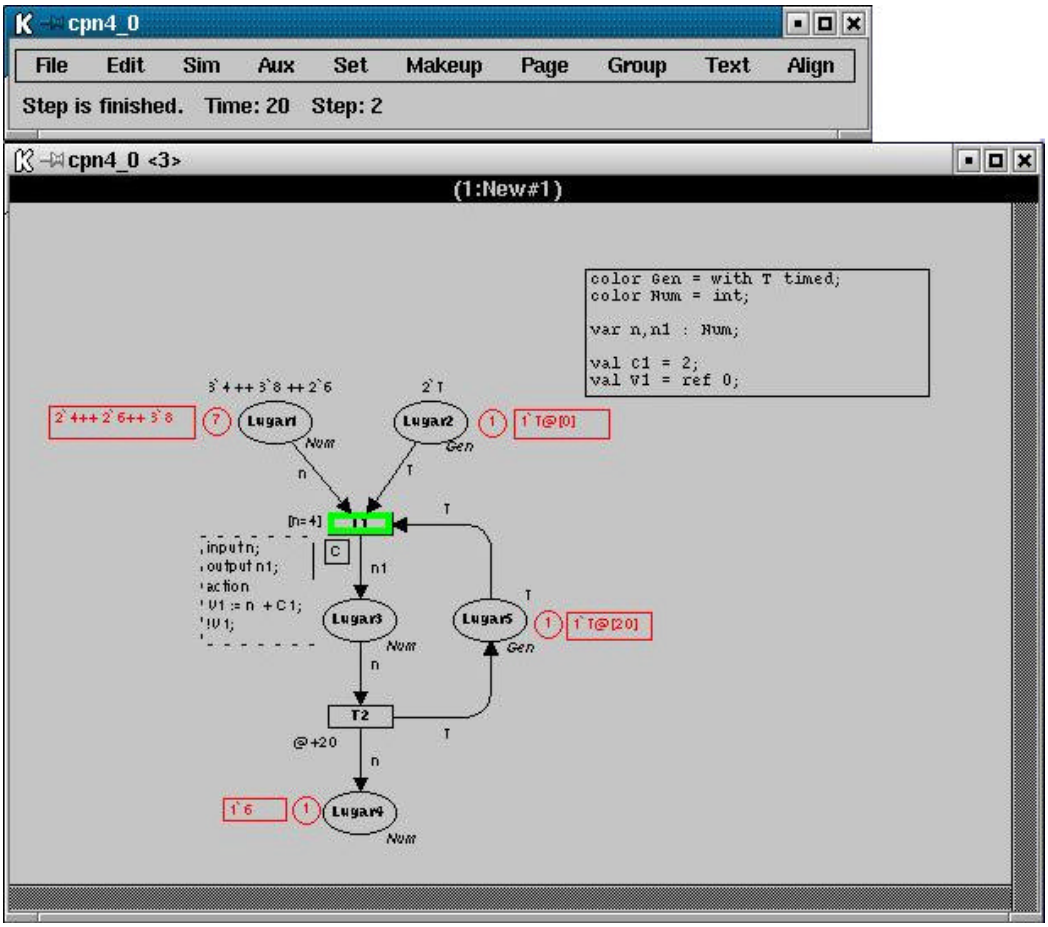
Cuando se realiza la primera ejecución, el sistema evoluciona al siguiente estado, el cual se observa en la figura 101. En este caso la transición *T1* deja de estar activa porque con su disparo el token de *Lugar5* desaparece. La transición activa en el paso 1 es la transición *T2*. Es de observar que hasta el momento el tiempo no ha avanzado.

Figura 101. Ejemplo: Segundo paso en la simulación



La figura 102 ilustra el tercer paso en la ejecución de la simulación. Para este paso, la transición *T2* ha introducido un retardo a los tokens resultantes de su ejecución. Si el simulador no encuentra más lugares activos en el tiempo de evaluación, inmediatamente se desplazará al tiempo de la siguiente activación. En este ejemplo, después de la activación de la transición *T2* en el tiempo cero (0), no hay más transiciones activas hasta que el tiempo de retardo sea cumplido, por lo tanto el simulador cambia hasta el tiempo veinte (20), como se puede observar en el la figura.

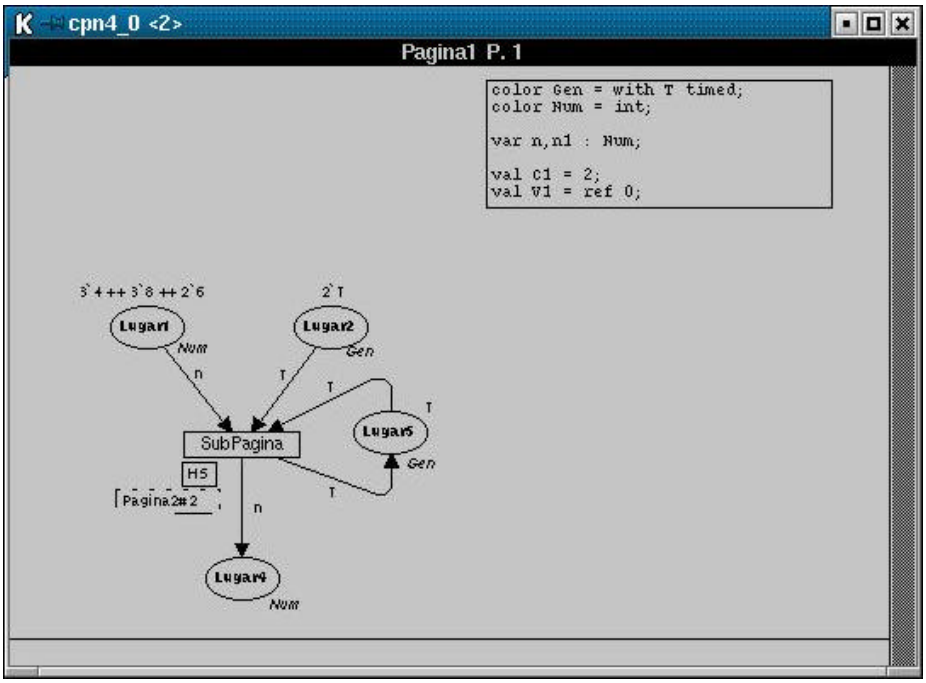
Figura 102. Ejemplo: Tercer paso en la simulación



Continuando con la simulación se podrán observar los estados siguientes hasta que la red llegue a un estado de bloqueo por el agotamiento de marcas en *Lugar2*.

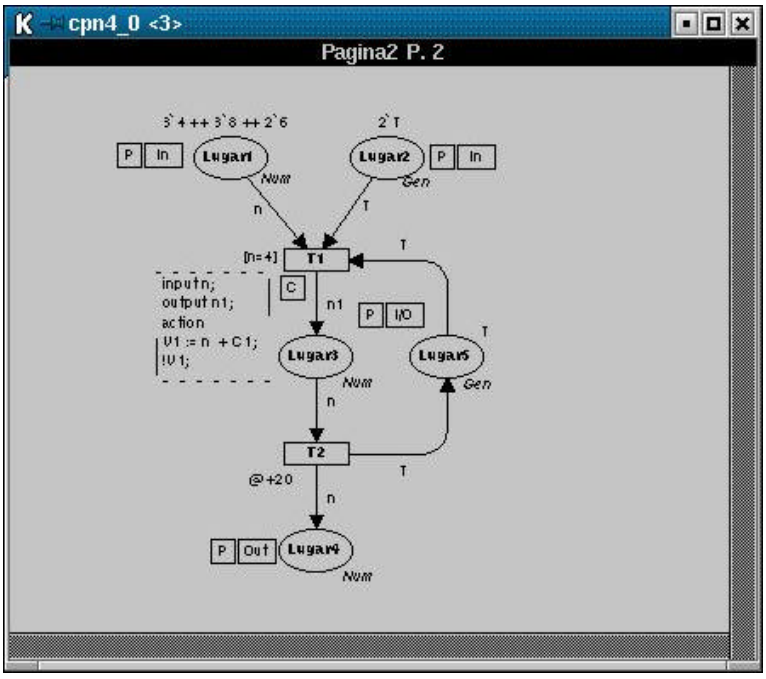
A continuación se implementará un nivel de jerarquía a la red de Petri de este ejemplo. Para este fin, se deben señalar a *T1*, *Lugar3*, *T2* y *Lugar5*, y utilizar la opción *Move to Subpage* del menú *CPN*. La figura 103 ilustra el resultado de este procedimiento.

Figura 103. Ejemplo: primer nivel jerárquico



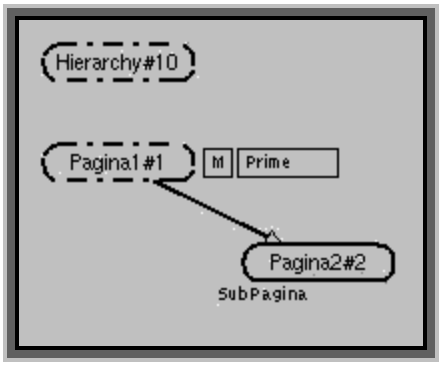
Internamente la red de Petri se observa como se ilustra en la figura 104.

Figura 104. Ejemplo: segundo nivel jerárquico



Finalmente, la figura 105 muestra la página de jerarquía para este modelo. Se han asignado los nombres a las páginas para diferenciarlas claramente: pagina1 y pagina2.

Figura 105. Ejemplo: Página de jerarquía para los dos niveles



OBSERVACIONES PARA EL USO DE DESIGN/CPN

- Los retardos se asignan anteponiendo "@+" al valor del retardo.

- La cantidad de tokens en un lugar se expresa utilizando " ´ " en medio del número de tokens y el tipo de token. No es válido utilizar " ' ", presenta errores de sintaxis.

- Para adicionar más de un token inicial a un lugar, se debe utilizar "++".

- El color *real* no es válido en la implementación de la red. Como valor o valor referenciado se puede utilizar, pero solo en las regiones de código.

- El teclado numérico debe estar desactivado para el funcionamiento normal del editor (recomendación por experiencia práctica propia).