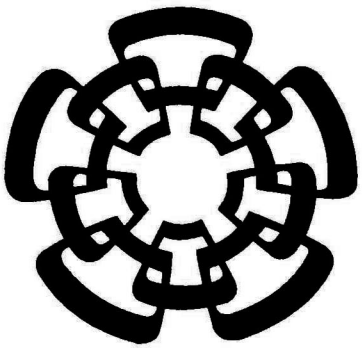
BC-640
Don.-2011

xx (179058.1)



CINVESTAV

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Una Herramienta para el Análisis de Modelos en redes de Petri

Tesis que presenta:

Ana Karina Ríos Araujo

para obtener el grado de:

Maestro en Ciencias

en la especialidad de:

Ingeniería Eléctrica

Directores de Tesis

Dr. Luis Ernesto López Mellado

Dr. Antonio Ramírez Treviño

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Agosto de 2010.

Una Herramienta para el Análisis de Modelos en redes de Petri

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:
Ana Karina Ríos Araujo
Ingeniero en Sistemas Computacionales
Instituto Tecnológico de Ciudad Guzmán 2002-2007

Becario del Conacyt, expediente no. 212762

Directores de Tesis
Dr. Luis Ernesto López Mellado
Dr Antonio Ramírez Treviño

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Agosto de 2010.

Índice general

Resumen

En la actualidad las redes de Petri (RP) son utilizadas ampliamente para especificar y analizar sistemas de eventos discretos (SED). Las ventajas del uso de este formalismo en cuanto a claridad y compacidad de los modelos obtenidos son grandes, sobre todo cuando se compara con las proporcionadas al utilizar autómatas finitos. Sin embargo cuando la talla del modelo es grande, es conveniente contar con facilidades computacionales que permitan automatizar la creación y el análisis de los modelos.

En este trabajo se aborda la problemática descrita mediante el desarrollo de una herramienta computacional para el análisis de modelos de SED expresados con RP. Esta herramienta, llamada SPADES, permite capturar visualmente un modelo a través de un editor gráfico; además numerosas técnicas de análisis pueden ser aplicadas al modelo invocándolas a través de menús. Una de las características más importantes de SPADES es su extensibilidad, ya que desde en el diseño se previó la facilidad de inclusión de nuevas funciones de análisis.

En la tesis se hace un breve análisis de herramientas de análisis existentes, se describe concisamente el proceso de ingeniería de software, los aspectos más relevantes de la implementación, y se incluye una guía para la extensión de la herramienta.

Abstract

Nowadays Petri nets (PN) are widely used for specifying and analyzing discrete event systems (DES). The advantages of using this formalism regarding clearness and compactness of the obtained models are numerous, especially when they are compared against that provided by finite automata. However when huge models are dealt it is convenient to have software facilities allowing automating the creation and analysis of models.

This work addresses the problem given above through the development of a software tool for analyzing models of DES expressed by PN. This tool, named SPADES, allows capturing visually a model through a graphical editor; also, many analysis techniques can be applied to the model by invoking them from menus. One of the most important features of SPADES is its extensibility, since from the tool design stage, the facility to include new functions has been foreseen. This thesis include an overview of existing analysis software tools, the software engineering process, relevant issues on the implementation, and a guide for extending the analysis tool.

Agradecimientos

A mi Madre por creer siempre en mí, por su inmenso amor, por ser mi mejor amiga, por las charlas interminables.
A mi Padre por su gran apoyo y sus palabras siempre tan acertadas.
A mis Hermanos por estar siempre ahí.
A mis Sobrinos por hacerme sonreír siempre.
A mis Abuelos por su cariño y sus oraciones.
A David por apoyarme incondicionalmente, por sus palabras de aliento, por su cariño, por estar a mi lado.
A mis asesores, el Dr. Luis Ernesto López Mellado por dejarme ser, el Dr. Antonio Ramírez Treviño por su gran apoyo, por sus consejos, por escucharme siempre.
A mis amigos.
A CONACYT, por el apoyo económico proporcionado.
A CINVESTAV Guadalajara por las facilidades y comodidades para el desarrollo de esta tesis.

Capítulo 1

Introducción

En la actualidad, muchos de los sistemas desarrollados por el hombre pertenecen a la clase de Sistemas de Eventos Discretos (SED), es decir aquellos sistemas con un número de estados numerable, donde el estado estable representando el desarrollo de una actividad cambia abruptamente con la ocurrencia de eventos. Muchos sistemas pueden incluirse dentro de esta clase, por ejemplo, algunos Sistemas de Manufactura Flexible (SMF) cuando consideramos a las piezas como numerables y a los límites de ejecución de procesos como eventos; el nivel de un tanque de agua, cuando se establecen umbrales para diferentes alturas del líquido; los Sistemas de Control de Tráfico Aéreo (SCTA), cuando las actividades se ven como el despegue y arribo de aviones, etc. Otras aplicaciones pueden ser encontradas en [5] y [12].

En los últimos años la complejidad de este tipo de sistemas se ha incrementado, requiriéndose cada vez más el uso de herramientas computacionales para el análisis y simulación de dichos sistemas. Aunque ya existen algunas herramientas, éstas carecen de algunas facilidades para analizar ciertas propiedades de los SED modernos. Por ejemplo algunas herramientas tienen interfaces no amigables, otras son cerradas o cuentan solo con un conjunto fijo de algoritmos de análisis.

Debido a la limitación en las herramienta existentes, este trabajo propone el desarrollo de la herramienta llamada SPADES (Specification and Analysis of Discrete Event Systems), la cual permite editar gráficamente un modelo expresado con redes de Petri(RP) y ofrece un conjunto de procedimientos para su análisis y simulación. Además se proporciona un conjunto de clases y métodos de acceso a datos para que el usuario pueda diseñar nuevos procedimientos de análisis y agregarlos a SPADES de una forma sencilla y transparente.

Antes de definir la herramienta SPADES, se presenta la definición formal de SED

y posteriormente se presenta el formalismo que se usará para representar SED.

Definición 1.1 *Los SED son aquellos que están compuestos por elementos que manejan entidades discretas, es decir, numerables y diferenciables entre si. Su funcionamiento está caracterizado por una sucesión finita o infinita de estados estables delimitados por eventos que ocurren, generalmente de manera asíncrona. [11]*

En la figura 1.1 se puede apreciar un ejemplo de un sistema de eventos discretos.

En el análisis de SED el primer paso es representar al sistema mediante algún formalismo de modelado, para posteriormente generar simulaciones y pruebas formales que lleven a ajustar el sistema para la eliminación de errores. Aplicar correcciones directamente sobre el sistema puede resultar costoso o de alto riesgo, por lo que experimentar con una representación de éste es una opción viable.

Existen distintas alternativas para la representación de SED como son las álgebras de procesos, expresiones regulares, los autómatas y las redes de Petri entre otros.

Las redes de Petri (propuestas por Carl Adam Petri en 1962) resultan ser la alternativa más atractiva debido a la simplicidad de su naturaleza gráfica, su soporte matemático simple, la claridad en su descripción y la facilidad para representar comportamientos complejos de sistemas que incluyan secuencias, concurrencia, paralelismo, sincronizaciones, intercambio de información y asignación de recursos entre otros.

Cuando se expresa un sistema mediante las redes de Petri, éstos puede ser validados mediante distintos análisis que permiten determinar propiedades estructurales, aquellas que se obtienen independientemente del marcado de la red, o de comportamiento, aquellas que si dependen del marcado inicial de la red.

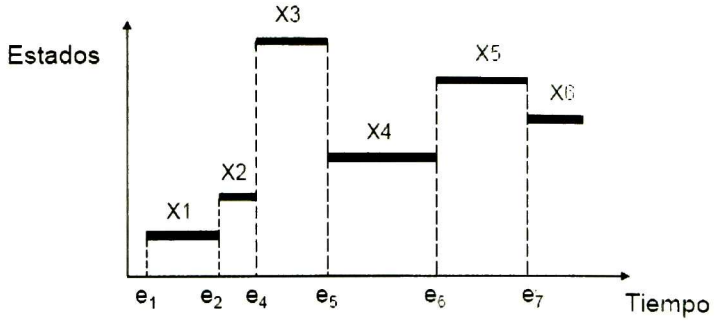


Figura 1.1: Ejemplo de un sistema de eventos discretos

Los sistemas de eventos discretos es un área que está en continuo desarrollo. Existen diversos grupos de investigación que trabajan en desarrollar nuevas técnicas de análisis que permiten optimizar los sistemas, conocer sus propiedades o analizar su desempeño. Por lo que constantemente se están generando nuevos algoritmos de análisis o mejorando los ya existentes. Lo que hace necesario contar con una herramienta que permita agregar estos nuevos procedimientos de una manera sencilla y obtener distintos tipos de estadísticas de análisis.

1.1. Herramientas computacionales existentes orientadas a redes de Petri

En la actualidad existen una gran variedad de herramientas computacionales que permiten la captura y análisis de sistemas de eventos discretos. Ellas utilizan algún método formal (álgebras de procesos, expresiones regulares, autómatas, redes de Petri...) para el análisis, y cada una de ellas posee diferentes características.

Las herramientas computacionales que nos interesa analizar son aquellas basadas en redes de Petri. Algunas de estas herramientas sólo permiten editar una red de Petri, a las cuales denominaremos *Editores*. Otras en las que además de la edición, también permiten la simulación del sistema y las nombraremos *Simuladores*, y algunas que van más allá y además de lo anterior permiten a los usuarios analizar la red de Petri capturada, mediante distintos procedimientos de análisis ofrecidos por la herramienta, a éstas últimas las llamaremos *Analizadores*.

1.1.1. Editores

Este conjunto de herramientas poseen herramientas de dibujo para la captura de la red de Petri en el editor. Algunas de ellas poseen una interfaz sencilla e intuitiva como CoopBuilder, FLOWer o LoLA. Incluso hay herramientas como Maria, MISS-RdP o Snoopy que permiten la captura de diferentes tipos de redes de Petri, como son las redes de Petri Coloreadas (RPC), redes de Petri Estocásticas (RPE) y redes de Petri Temporizadas (RPT), respectivamente.

Otras más están realizadas en lenguajes como Java CoopnBuilder, JARP. Algunas sólo pueden ser ejecutadas en plataformas específicas ya sea en Windows o Linux PED, VisualPetri.

El editor de la herramienta SPADES que se propone, debe poseer una interfaz

elegante y más intuitiva que las de este conjunto de herramientas. Los elementos de dibujo (nodos, arcos, selección) deben de estar dispuestos en forma de una barra de herramientas y en una posición visible para el usuario, además de ser sencillo introducir los elementos al editor y tener una representación más atractiva. Aunque SPADES no debe poseer de manera inicial la representación de diferentes tipos de redes de Petri, la herramienta estará abierta para que el usuario genere las extensiones que requiera.

En la tabla 1.1 se presenta un conjunto de herramientas y algunas de sus características.

Tabla 1.1: Editores

Nombre	**RP que soporta**	**Plataforma**
ALPHA/Sim	HLPN, TPN	Sun, Solaris, Windows
CoopnBuilder	OOPN	Java
CPN Tools	HLPN, TPN	Linux, Windows
HISIm	HLPN, PTPN, TPN	Java
MISS-RdP	SPN, TPN, CPN	Windows, Sun
PED	PTPN	Sun, Linux
PetriSim	HLPN, PTPN, TPN	MS DOS
Snoopy	SPN, PTPN, TPN	Linux, Windows, Mac OS
VisualPetri	PTPN	Windows
Helena	HLPN	Linux
COSA BPM	HLPN	SunOS, HP, Linux, Windows
SEA	HLPN, TPN	Sun

1.1.2. Simuladores

En algunas de estas herramientas, no se aprecia de manera clara la evolución de la red mientras es simulada, tal es el caso de ARP, HiQPN-Tool, mientras que algunas otras sólo permiten la simulación manual.

F-net, ARP ejecutan de manera amigable la simulación, sin embargo el manejo de su interfaz no resulta agradable por lo que resulta tedioso su utilización.

SPADES debe proveer al usuario la opción de ejecutar la simulación de manera manual, es decir el usuario debe decidir cuál de las transiciones habilitadas debe ser

disparada, y automática, es decir el sistema de manera aleatoria decide la transición a disparar.

Cuando se ejecute la simulación, los controles que modifican la red se deben deshabilitar para mantener la consistencia de la red. Así mismo, las transiciones habilitadas deben ser coloreadas para distinguirlas de aquellas que no lo estén, también debe ser fácil observar el paso de marcas de un lugar a otro.

En la Tabla 1.2 se muestran todas las herramientas de esta categoría.

Tabla 1.2: Simuladores

Nombre	**RP que soporta**	**Plataforma**
ARP	PTPN, TPN	MSDos
Artifex	OOPN, TPN, HLPN, PTPN	Sun, Linux, IRIX, Windows
CPN-AMI	HLPN, PTPN	Linux, Mac Os
F-net	SPN, TPN	Windows
GreatSPN	HLPN, SPN, TPN	Linux, Sun
HiQPN-Tool	HLPN, SPN	Sun
INA	HLPN, SPN, PTPN	Sun, Linux, Windows
PROTOS	PTPN, SPN, TPN	Windows
TimeNET	HLPN, PTPN, SPN, TPN	Linux, Windows

1.1.3. Analizadores

La principal diferencia entre la mayoría de las herramientas de este conjunto y SPADES, es que las primeras sólo proveen un conjunto fijo de procedimientos de análisis que pueden ser aplicados y no son sistemas abiertos. Ejemplos de estas herramientas de análisis son Great-SPN, JFern, ExSpect.

Algunas de estas herramientas no provén una visualización agradable de los resultados arrojados por los análisis. Mientras que SAPDES debe presentar sus resultados en ventanas independientes al editor, con los datos que el usuario eligió visualizar, y en el acomodo que le pareció más agradable para presentarlos.

En la Tabla 1.3 se muestra un conjunto de herramientas que poseen estas características, pero además existen otras tres herramientas, Maria, Predator y PIPE2 que permiten agregar procedimientos de análisis de manera dinámica, característica

importante de nuestra herramienta SPADES. A continuación se describen cada una de ellas.

Tabla 1.3: Analizadores

Nombre	RP que soporta	Plataforma
Maria	HLPN, PTPN	Sun, UNIX, IRIX, HP-UX, Linux, Mac Os, Windows
Great-SPN	HLPN, SPN, TPN	Sun, Linux
ExSpect	HLPN, TPN	Linux, Windows
Great-SPN	HLPN, PTPN, TPN	Sun, Linux
PIPE2	TPN, CPN	Windows, Java
Predator	PTPN, SPN	Java
JFern	HLPN, PTPN, TPN	Java

María

María es un analizador de alcanzabilidad para sistemas concurrentes que utilizan sistemas de redes algebraicas (una variante de redes de Petri de alto nivel) como su formalismo de representación. María incluye un algoritmo distribuido para la verificación de propiedades. Dicho algoritmo puede ser ejecutado en un equipo con múltiples procesadores así como en estaciones de trabajo sobre redes TCP/IP. Gracias a su diseño modular y licencia de código abierto, María puede ser fácilmente extendido con nuevos algoritmos. Su interfaz gráfica está basada en GraphViz.

PIPE

PIPE (Plataform Independent Petri Net Editor) es una herramienta que permite la captura de redes de Petri ordinarias y temporizadas, permite la simulación de las mismas tanto de manera manual como automática. PIPE Utiliza el estándar PNML (Petri Net Markup Lenguage) el cual es un formato de intercambio entre aplicaciones basado en XML, que busca la Flexibilidad, compatibilidad y no ambigüedad en la representación de las redes de Petri. PIPE fue desarrollado por el Departamento de Cómputo del Colegio Imperial de Londres en el año de 2002 y ha ido creciendo en funcionalidad ya que es de código abierto y está implementado en Java.

Predator

Predator fue desarrollado para proveer una eficiente manera de diseñar y mostrar redes de Petri jerárquicas, mediante el uso del concepto de subredes. Predator permite a sus usuarios cargar de manera dinámica módulos de análisis. Dichos módulos se implementan como una simple interfaz java. Actualmente la herramienta cuenta con la implementación de un modulo de análisis que permite la obtención de los invariantes.

Si se desea profundizar en el estudio de las herramientas aquí presentadas, un lector interesado puede consultar [10]

1.1.4. SPADES versión previa

SPADES versión previa (SPecification Analysis of Discret Event Systems) es una herramienta que posee un editor mediante el cual puede introducirse la red de Petri. Permite la captura de redes de Petri ordinarias, Coloreadas, Interpretadas, Estocásticas y Temporizadas. Permite la simulación tanto manual como automática de la red, así como también provee de un conjunto de procedimientos limitado con los que el usuario puede obtener distintos tipos de análisis. Dicho conjunto de procedimientos puede apreciarse en las Tablas 1.4 y 1.5.

Tabla 1.4: Procedimientos de análisis estructurales y de determinación de propiedades ofrecidos por SPADES-V1.

Análisis Estructurales	**Determinación de Propiedades**
S y T Invariantes	Acotabilidad
Sifones y trampas	Conservatividad
Componentes conexas	Libre de bloqueo
Componentes fuertemente conexas	class
Home Spaces	Evento detectabilidad
Distancia sincrónica	Lugares implícitos
	Vivacidad
	Repetitividad
	Free-Choice bien formadas

Por conveniencia nombraremos a SPADES versión previa como *SPADES-V1* y al SPADES desarrollado en esta tesis como *SPADES-V2.*

Tabla 1.5: Procedimientos para el análisis Dinámicos y de Evaluación de Desempeño que ofrece SPADES-V1.

Propiedades Dinámicas	Evaluación de Desempeño
Observabilidad	Tiempo de ciclo
Controlabilidad	Grafo de estados diligente
Alcanzabilidad	

A pesar de que SPADES-V1 posee un número considerable de procedimientos de análisis, no es una herramienta que permita al usuario adicionar aquellos procedimientos que sean generados por él mismo, o aquellos procedimientos que generen los grupos de investigación del área, por lo que limita la funcionalidad de la herramienta afectando la usabilidad de la misma a largo plazo, convirtiéndola, en un momento dado, en una herramienta obsoleta. Así mismo el editor gráfico que proporciona SPADES-V1 al usuario para introducir la red de Petri, no es muy intuitivo, se dificulta la manipulación de los elementos de la red de Petri, y la modificación de los atributos de lugares, transiciones y arcos es un tanto complicada.

La herramienta SPADES-V1, no es considerado como un trabajo previo a SPADES-V2, ya que SPADES-V2 fue creado desde cero, no se tomó ningún módulo o parte del código de SPADES-V1 para el desarrollo de SPADES-V2.

1.2. Metodología de desarrollo de SPADES

Para desarrollar este tipo de herramientas de una manera adecuada se hace necesario el utilizar técnicas de la ingeniería de software que ayuden al desarrollador durante las etapas del diseño del software, las cuales comprenden desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de éste después de que se utiliza.

Es bien conocido, como usuario de cualquier herramienta computacional, cuan frustrante es utilizar un producto de software que no permite desempeñar las tareas básicas para las cuales fue desarrollado. De igual manera, para los desarrolladores es frustrante captar la funcionalidad que el usuario esperaba sólo después de que ya se ha implementado el sistema. Por lo que es necesario definir el conjunto de requerimientos que formarán la funcionalidad del sistema y que permitan establecer un acuerdo entre

el desarrollador y el cliente.

"Los requerimientos son... una especificación de lo que debe ser implementado. Son descripciones de cómo el sistema debe comportarse, un atributo o alguna propiedad del sistema. Incluso estos pueden ser restricciones en el proceso del desarrollo del sistema" [26].

Es decir, los requerimientos describen lo que se obtendrá cuando el proyecto haya concluido. Hacen referencia a los requerimientos funcionales y al comportamiento del sistema que será exhibido bajo ciertas circunstancias. Los requerimientos pueden expresar restricciones en el proceso de desarrollo o en su funcionamiento. Pueden expresar propiedades del sistema o atributos, es decir, lo que el sistema hace y cómo lo hace.

Muchos problemas del software se derivan de deficiencias en las formas en que se documenta, realizan acuerdos, y modifica los requerimientos del producto. Las áreas problemáticas podrían incluir la recopilación informal de la información, funcionalidad implícita, suposiciones erróneas, requerimientos definidos de forma inadecuada, y un cambio casual durante el proceso de desarrollo.

Entre el cuarenta y sesenta por ciento de los errores o defectos en un proyecto de software se realizan durante la etapa de captura de requerimientos, específicamente, en la gestión y la especificación de estos [25]. El resultado típico es una diferencia de expectativas: lo que los desarrolladores piensan que deben de construir y lo que los clientes realmente necesitan. Por lo que la ingeniería de requerimientos es uno de los aspectos más desafiantes del desarrollo de software. Es posiblemente el paso más importante, ya que sienta todas las bases para el trabajo subsecuente del proyecto.

Aunque por décadas los diseñadores de software han construido sistemas basados exclusivamente en los requerimientos técnicos. Los requerimientos hacen explícito algunas, y sólo algunas, de las propiedades deseables del sistema final; por lo que puede decirse entonces que la especificación de los requerimientos sólo comienzan a contar la historia del software que se requiere desarrollar. [2]

En el desarrollo de software actual, la arquitectura del sistema forma parte central de un buen diseño. La arquitectura conjunta la definición previa del sistema, a la vez que permite una visión global de la estructura del sistema en una etapa temprana en el desarrollo del mismo.

Se puede afirmar que la arquitectura no existe por si misma, es parte de un ciclo. Es influenciada por los requerimientos funcionales del sistema y las propiedades de calidad, el cliente y la organización de desarrollo, por la experiencia del arquitecto y por todos aquellos interesados en la construcción de sistema [2].

Para conocer más acerca de la importancia y el ciclo de desarrollo de la arquitectura consulte [24], [21], [19], [2].

En [22] el diseño de software se centra en la definición de la funcionalidad del sistema a través de la división de los requerimientos, identificación de subsistemas, asignación de requerimientos a los subsistemas, especificación de la funcionalidad y definición de las interfaces de los subsistemas. Este conjunto de actividades, así como la definición de los requerimientos pueden ser modelados como un conjunto de componentes y de relaciones entre estos componentes, ilustrándose gráficamente como un modelo arquitectónico. Dicho modelo arquitectónico representará la división y clasificación de los requerimientos mediante grupos que describen o especifican una funcionalidad que sera ofrecida por el sistema, dando así, origen a los diferentes subsistemas que conformarán la herramienta en su totalidad.

El diseño de la arquitectura es influenciada por todos aquellas personas que de alguna manera están interesados o involucrados en la construcción del sistema de software. Estas personas son conocidos como stakeholders y pueden ser el cliente, el usuario final, el analista, los programadores, etc. El problema es que cada uno de ellos tiene una diferente meta, funcionalidad o propiedad que les gustaría que el sistema garantizara u optimizara, lo que puede resultar en contradicciones. Por lo que el arquitecto de software debe llenar los espacios en blanco o en su caso mediar entre los posibles conflictos que existen entre los requerimientos. Incluso la escasa experiencia del arquitecto pueden llevar al diseño de una arquitectura no adecuada o pobre.

Por lo que en ocasiones resulta útil que el arquitecto de software se apoye en los esquemas arquitectónicos existentes que describen propiedades de calidad de acuerdo al sistema que se modela. Los patrones arquitectónicos es unos de los puntos en donde la comunidad de arquitectura de software ha llegado a un consenso: su trascendencia es notoria y son esenciales en la concepción de una arquitectura. [21]

El desarrollo de sistemas de software lleva consigo todo un ciclo de producción que permitirá realizar un sistema apegado a las necesidades del cliente. Este ciclo comprende varias etapas que permiten dar forma al sistema que se diseña. Especificar, diseñar, implementar, validar y mantener el sistema son las etapas que comúnmente son ejecutadas por los desarrolladores del sistema.

Los servicios que debe proporcionar el sistema, las restricciones sobre las que debe ser construido el sistema y sobre las cuales debe funcionar, las formas en que el sistema puede ser usado para cumplir su propósito, etc. son algunos ejemplos de los elementos a considerar para generar el sistema.

De manera general las etapas pueden ser definidas de la siguiente manera.

Especificación

Se describe lo que el sistema debe hacer (sus funciones) y sus propiedades esenciales y deseables mediante consultas al cliente y/o usuario final del sistema. Una parte importante de esta fase es establecer un conjunto completo de objetivos que el sistema debe cumplir. Éstos no necesariamente deben expresarse forzosamente en términos de la funcionalidad del sistema, pero deben expresar por qué se construye el sistema.

Diseño

Se centra en proporcionar la funcionalidad del sistema a través de la creación de distintos componentes que describen los diferentes subsistemas que conformaran la herramienta en su totalidad. Esto se logra mediante la división y agrupación de requerimientos en base a funcionalidades especificas. El diseño depende de la especificación, debido a que hace uso de los requerimientos recopilados en esta etapa para definir los subsistemas. Éstos a su vez se modelan como un conjunto de componentes y relaciones entre dichos componentes ilustrándose gráficamente como el modelo arquitectónico del sistema. Aunque más adelante, se podrá ver que lo anterior puede resultar insuficiente.

Implementación

El conjunto de susbsistemas que se hayan identificado durante la etapa de diseño pasan a ser implementados. Lo que implica generalmente analizar si el susbsistema no requiere mayor especificación en sus funcionalidades, o si debe ser particionado para un mejor manejo y posterior control del mismo. Estas revisiones comúnmente implican cambios en el software por lo que es importante diseñar software para el cambio de modo que puedan ser implementados nuevos requerimientos sin excesivo coste adicional. Posteriormente estos subsistemas se conjuntan para formar el sistema completo.

Validación y Mantenimiento

La validación se aplica a cada subsistema en el que se valora la calidad con la que proporciona la funcionalidad para lo cual fue creado. Así mismo comprende el aplicar un conjunto de pruebas mediante las cuales se debe comprobar las conexiones entre cada componente y el comportamiento en su totalidad.

El mantenimiento se refiere a la corrección de errores en los requerimientos originales e implementar las correcciones de aquellos nuevos errores que surjan durante el desarrollo del sistema. Muchos de los sistemas desarrollados suelen depender de tecnologías de hardware o software que con el paso del tiempo se vuelven obsoletas y se necesario modificar el sistema.

1.3. Objetivos de la tesis

- Desarrollar una herramienta de cómputo que facilite la expresión de modelos en redes de Petri a través de un editor gráfico.
- Permitir la simulación del modelo.
- Permitir ejecutar procedimientos de análisis sobre el modelo.
- Facilitar la extensión de las capacidades de análisis de la herramienta.
- Capacidad de ejecución en distintas plataformas.

La tesis está organizada de la siguiente manera:

El Capítulo 2 describe el conjunto de requerimientos de usuario, los requerimientos del sistema y los requerimientos no funcionales que forman la especificación del sistema SPADES, así mismo se muestran los diagramas UML de caso de uso y de secuencia que representan el comportamiento del sistema.

El Capítulo 3 introduce brevemente los distintos patrones arquitectónicos, i.e. esquemas que utilizan los ingenieros de software para diseñar arquitecturas. Debido a las características del patrón MVC, es el que se selecciona para diseñar de la arquitectura del sistema SPADES.

En el Capítulo 4 se hace la presentación de los menús que contiene SPADES, y se describen las funcionalidades que ofrece el sistema, como son, la edición de los atributos de los elementos de la red de Petri, la forma en que el usuario puede introducir su modelo y de qué manera puede simular su funcionamiento. También se describe cómo el usuario puede agregar y eliminar procedimientos de análisis, y cómo generar una nueva extensión de red de Petri.

Por último, en el Capítulo 5 se presentan las Conclusiones y el Trabajo futuro.

Capítulo 2

Especificación de la Herramienta

En este capítulo se presenta el conjunto de requerimientos que conforma la especificación del sistema. Dichos requerimientos describen las necesidades del usuario, las características funcionales, las propiedades de calidad y restricciones a las cuales está sujeto el sistema.

Existe una variada clasificación de requerimientos que describen diferentes niveles de especificación del sistema. En esta tesis sólo se expresan dos de ellos. Los *requerimientos de usuario* y los *requerimientos del sistema.*

Los *requerimientos de usuario* deben describir los requerimientos funcionales del sistema de tal forma que sean comprensibles por los usuarios del sistema sin conocimiento técnico detallado. Únicamente deben especificar el comportamiento externo del sistema y deben evitar, tanto como sea posible, las características de diseño del sistema. Por consiguiente, no se debe utilizar lenguaje técnico o notaciones formales, sino que deben ser redactados en un lenguaje sencillo [22].

Los *requerimientos del sistema* por su parte, son versiones extendidas de los requerimientos del usuario que son utilizados por los ingenieros de software como punto de partida para el diseño del sistema. Agregan detalle y explican cómo el sistema debe proporcionar los requerimientos del usuario, por lo que deben ser una especificación completa y consistente del sistema entero a la vez que describen las restricciones operativas del sistema [22].

2.1. Definición de requerimientos del sistema SPADES

A continuación se presenta una lista que especifica los *requerimientos de usuario* y los *requerimientos del sistema*, los cuales se disponen de la siguiente manera: en negritas se describe el requerimiento de usuario, enseguida el conjunto de requerimientos del sistema correspondiente a dicho requerimiento de usuario y así sucesivamente.

- **La herramienta debe permitir al usuario introducir gráficamente su sistema mediante una red de Petri**

  a La herramienta debe contar con un editor gráfico en el que el usuario pueda dibujar una red de Petri.

  b Para poder dibujar la red, el usuario debe hacer uso de las *herramientas de dibujo*. Estas herramientas de dibujo contendrá *lugares*, *transiciones* y *arcos*.

  c El editor gráfico contará con una cuadrícula que ayude al usuario a dibujar su modelo con mayor estética.

  d Las herramientas de dibujo deben estar a simple vista del usuario sin que éste tenga que buscarlas en algún menú o submenú, para lo cual se dispondrán en una barra general contenedora.

  e Cuando el usuario dibuje su modelo, sólo debe seleccionar la *herramienta de dibujo lugar*, *transición* o *arco* según desee, y cuantas veces lo desee y colocar el cursor sobre la cuadrícula del editor, para que éstos sean agregados.

  f El sistema no debe permitir relacionar un par de nodos iguales, es decir, lugar con lugar o transición con transición.

  g Los lugares y transiciones deben de contar con un identificador que los distinga de sus similares.

La herramienta debe permitir introducir, mover, colocar, eliminar, arrastrar y seleccionar elementos de la red de una manera sencilla e intuitiva.

a Cuando el usuario desee mover, eliminar, arrastrar o seleccionar la totalidad de la red o elementos de la misma, debe hacer uso de la *herramienta de selección.*

b Mediante la *herramienta de selección*, el usuario podrá mover los elementos de la red de lugar sin que éstas se desconecten.

c La *herramienta de selección* debe estar dispuesta en la barra general contenedora a la vista del usuario.

d Para eliminar uno o varios elementos de la red modelada, el usuario sólo debe seleccionar el elemento y presionar suprimir en su teclado.

e Cuando un nodo (lugar o transición) sea movido de lugar y esté unido mediante uno o varios arcos, el arco deberá ajustar su tamaño a la nueva posición del nodo.

f El arco podrá dibujarse como una linea continua y angular en distintos puntos para permitir un mejor acomodo del mismo en el dibujo.

g Las transiciones podrán girarse en distintos ángulos.

h Las etiquetas de los nodos podrán ser cambiadas de posición a la ofrecida por defecto inicialmente.

- **SPADES debe proporcionar funcionalidades de manipulación de archivo.**

a El usuario puede almacenar en disco el archivo mediante la opción *guardar*, abrir uno existente seleccionando la opción *abrir*, renombrarlo mediante *guardar como* o cerrar el archivo que actualmente se manipula mediante la opción *cerrar.*

b Todas las opciones mencionadas en el punto anterior deben estar contenidas en un menú denominado *Archivo.*

c Para las operaciones de guardado o apertura de un archivo, el sistema debe mostrar una *ventana de archivos* la cual funcionará de la siguiente manera:

3.a Para abrir un archivo el usuario podrá buscarlo en cualquier directorio mediante esta ventana, una vez localizado, se selecciona el archivo y se elije *abrir* para visualizar el archivo, o *cancelar* para anular la acción de abrir.

3.b Para guardar un archivo por primera vez o si se eligió la opción *guardar como*, el usuario debe elegir la ruta en la que desee almacenar el archivo mediante esta ventana, indicar el nombre en el espacio proporcionado y seleccionar guardar, o cancelar para anular la acción de guardar.

d Cuando el usuario cierre el archivo que actualmente manipula, la aplicación debe preguntarle si desea almacenar los cambios no guardados, si el usuario acepta y en el caso de no haber almacenado nunca el archivo, aparecerá la ventana de archivos para que se le asigne un nombre y se cerrará el archivo. Si ya lo almacenó al menos una vez, el sistema guardará cambios sin mostrar la ventana de archivos y cerrará dicho archivo.

e Se puede renombrar un archivo tantas veces como el usuario lo desee.

f El editor debe permitir mantener abierto mas de un archivo a la vez.

- **La herramienta debe permitir la aplicación de procedimientos de análisis.**

  a El sistema debe contar con un conjunto de procedimientos que permita analizar la red que el usuario modela.

  b Este conjunto de análisis deberá de estar contenido en un menú denominado *procedimientos.*

  c Para aplicar alguno de dichos análisis el usuario sólo debe presionar el botón derecho del mouse (dar click) sobre el que éste desee.

  d Cuando el usuario ejecute un análisis y no exista una red en le editor, el sistema debe comunicarlo al usuario.

  e Los procedimientos serán almacenados en una carpeta específica dentro del sistema SPADES.

  f El código de cada procedimiento debe ser accesible al usuario cuando éste lo desee.

  g Los procedimientos estarán en su código fuente así como en su parte ejecutable.

- **Se debe permitir la exportación del sistema capturado por el usuario en formatos que no sean susceptibles a la degradación de la imagen.**

  a La red podrá ser exportada como imagen en formato .png o .ps.

b Las opciones para exportar el dibujo plasmado por el usuario como una imagen, deberán estar dispuestas en un menú.

c Cuando el usuario elija cualquiera de las dos opciones, se presentará la ventana de archivo para que el usuario indique un nombre y se elija la ruta de almacenamiento.

d La imagen exportada no podrá ser editada o manipulada en SPADES.

e La cuadrícula que ofrece como guía el editor, no aparecerá en la imagen exportada.

- **El usuario debe poder modificar los atributos de la red que modela mediante una interfaz sencilla.**

a Cada elemento de la red (lugares, transiciones, y arcos) deberá presentar un submenú que permita la modificación de atributos específicos.

b Dicho submenú se activará presionando el botón derecho del mouse sobre el elemento a editar (lo que seleccionará el elemento) y posteriormente presionando el botón izquierdo (secundario) sobre el elemento seleccionado.

c Las modificaciones hechas mediante el menú de atributos, sólo afectará al elemento seleccionado.

d Mediante el submenú referente a los lugares se podrá modificar el nombre de la etiqueta y número de marcas.

e En el submenú referente a transiciones se permitirá la modificación del nombre de la etiqueta así como el giro en grados de la transición.

f Cuando exista la relación transición-lugar, lugar-transición representada mediante dos arcos, cada uno en dirección contraria al otro, el submenú de los arcos permitirá unirlos para representar lo mismo pero con un solo arco de dos puntas.

g Así mismo, el submenú de transiciones permitirá modificar el peso de cada arco.

h Cada submenú contará con dos opciones, una que guarde los cambios efectuados y otra en la que se ignoren y se continúe sin modificación alguna.

- **La herramienta debe permitir la simulación del modelo.**

a La herramienta presentará un conjunto de herramientas de simulación, que permitan iniciar la simulación del modelo así como detenerla.

b Así mismo, las herramientas de simulación contendrán una opción que permita simular la red de manera automática y otra de manera manual.

c Cuando el sistema esté simulando los menús o herramientas que modifiquen la apariencia o atributos de la red, deberán de desactivarse, para evitar inconsistencias.

d Durante la ejecución de la simulación, manual o automática, las transiciones habilitadas se colorearán de un color distinto para que sean identificadas por el usuario.

e En la simulación automática el sistema ejecutará las transiciones habilitadas de manera aleatoria.

f Por su parte, en la simulación manual, el usuario ejecutará la transición habilitada que sea de su elección.

g Cuando se dispare una transición, de manera manual por el usuario o automática por el sistema, el paso de marcas de un lugar a otro, deberá ser percibible para el usuario.

h Cuando la simulación sea detenida, el marcado de la red será el inicial.

i La simulación deberá presentar los resultados de la corrida.

j Cuando la simulación sea detenida, los menús o barras de herramientas que se hayan deshabilitado, deberán volver a presentarse habilitadas..

- **El usuario debe poder observar la información que generen los procedimientos de análisis.**

a Cada procedimiento de análisis deberá presentar los resultados formateados por el usuario en una ventana independiente al editor, para que sean rápidamente visualizados.

b Los resultados de igual manera serán almacenados en un archivo, por si el usuario los requiere posteriormente.

La herramienta debe permitir agregar nuevos procedimientos de análisis.

a Para agregar el nuevo procedimiento el sistema contará con una plantilla estandard que ofrezca las librerías necesarias para su implementación.

b Cada procedimiento nuevo agregado será desplegado junto con los anteriores en el menú de procedimientos.

c La documentación del procedimiento dependerá del usuario que lo genere.

d El sistema debe proveer de un conjunto de métodos que permitan obtener los datos de la red que se haya capturado.

e El usuario debe implementar su procedimiento a manera que éste pueda ser reutilizado.

f La eliminación de un procedimiento debe de ser tan sencilla como borrar el archivo correspondiente.

- **El sistema debe modelar redes de Petri generalizadas así como coloreadas y temporizadas.**

a De manera transparente para el usuario, se debe crear la representación de cada uno de los elementos de la red de Petri.

b Cada elemento de la red de Petri debe tener atributos bien definidos.

c Para generar extensiones nuevas, el usuario sólo debe especificar nuevos atributos de la red de Petri.

d El usuario debe generar los métodos necesarios que le permitan obtener, modificar y/o visualizar los nuevos atributos o datos referente a la nueva extensión.

e Estas nuevas extensiones deberán de poder ser dibujadas por el usuario.

Este conjunto de requerimientos expresa la funcionalidad requerida para el sistema SPADES. Cada punto define los atributos deseados. Sin embargo, no expresa las *propiedades de calidad del software*, las cuales son igualmente importantes, ya que el incumplimiento de ellas puede ocasionar que el sistema sea tan problemático, como si funcionara de manera incorrecta.

2.2. Propiedades de Calidad existentes

Los atributos de calidad de un sistema han sido de gran interés en la comunidad de software al menos desde 1970.[2]. Los requerimientos de calidad, ayudan a especificar

la respuesta del software para llevar acabo los objetivos establecidos. Son decisiones de diseño que influyen en el control de respuesta del sistema para una propiedad dada.

Las propiedades de calidad son a menudo conocidos como los requerimientos no funcionales de un sistema, ya que no se refieren de manera directa a las funciones específicas que proporcionara el sistema sino a las propiedades emergentes de éste, como la disponibilidad, seguridad, rendimiento entre otros.

Esto significa que a menudo son mas críticos que los requerimientos funcionales particulares. Los usuarios del sistema normalmente pueden encontrar formas de trabajar alrededor de una función del sistema que realmente no cumple sus necesidades. Sin embargo el incumplimiento de un requerimiento no funcional puede ocasionar que el sistema entero sea inutilizable [22].

Como por ejemplo: en los sistemas distribuidos la ***disponibilidad*** es una propiedad deseable debido a las fallas o faltas que pueden ocurrir en el sistema, es decir, cuando el sistema ya no proporciona más un servicio correspondiente a su especificación o cuando una falta o la combinación de ellas causan que el sistema deje de funcionar por completo. En su caso, la propiedad de la disponibilidad aporta técnicas que ayudan a la recuperación del sistema ya sea mediante algún tipo de redundancia (Fault prevention), mediante algún tipo de monitoreo (Fault detection) o a través de algún tipo de recuperación cuando la falla es detectada (Fault recovery).

Supongamos el caso de un sistema web financiero, donde el número de peticiones de sus usuarios posiblemente puede superar las 5,000 consultas (Resource demand), la propiedad de ***rendimiento*** juega un papel importante, debido a que es necesario establecer cuanto tiempo le toma al sistema responder a dichas peticiones (Resource management), así como cuantas de éstas puede procesar por unidad de tiempo que se desee establecer (Resource arbitration). El objetivo de esta propiedad es generar una respuesta al evento o flujo de eventos que arriban al sistema con algunas restricciones de tiempo.

Usando el mismo sistema web financiero anterior como ejemplo, podemos establecer otra propiedad de calidad, la ***seguridad***. En este caso, el sistema debe ser capaz de detectar el intento de uso no autorizado (Detecting attacks), resistir al ataque del mismo (Resisting attacks) mientras se continua ofreciendo el servicio a aquellos usuarios permitidos. Así mismo el sistema debe ser capaz de recuperarse después de un ataque dado. El ataque puede ser un intento no autorizado por acceder a datos o servicios, la modificación de datos no permitidos e incluso un intento por denegar el servicio a usuarios legítimos.

Como se pudo observar un mismo sistema puede requerir cumplir con más de una

propiedad de calidad, ello dependerá del tipo de sistema que se diseñe, así como los requerimientos de calidad que se deseen satisfacer. Comúnmente estas propiedades son denominadas *propiedades emergentes.* Debido a que surgen del ambiente en el que el sistema se desempeña, de los sistemas con los que interactua, o incluso de las necesidades de sistemas terceros; es decir, las propiedades emergentes son requerimientos del sistema que no precisamente son los requerimientos que describen la funcionalidad del sistema deseada por el cliente.

Así como estos ejemplos son muchas más las propiedades emergentes de los sistemas, y como se pudo observar un mismo sistema puede requerir de más de una propiedad, las cuales dependerán del tipo de sistema que se diseñe así como la calidad con que se requiera generar el mismo.

2.3. Objetivos de Calidad para el sistema SPADES

Anteriormente se definieron los requerimientos funcionales del sistema SPADES, ahora hace falta establecer el conjunto de propiedades de calidad que se desean cumplir. Estas propiedades están estrechamente relacionadas con el diseño de la arquitectura, de hecho estas propiedades ayudan a determinar en un momento dado el diseño de la misma. Dicha relación se podrá ver en el capítulo correspondiente a la arquitectura.

La ***Modificabilidad*** es una propiedad deseable debido a que su objetivo es la reducción del tiempo para implementar cambios en un sistema, mediante el uso de tácticas como la localización de modificaciones (Localize modifications) y la prevención del efecto dominó (Prevent the ripple effect). Ambos aspectos permitirán al usuario adicionar al sistema sus propios procedimientos.

La localización de modificaciones permitirá asignar responsabilidades a los módulos durante la etapa de diseño del sistema, de tal manera, que los cambios o modificaciones previstas tendrán un alcance controlado. Esta táctica consiste en establecer la coherencia semántica entre las responsabilidades de cada módulo del sistema, tratando de garantizar que todas las responsabilidades trabajen juntas sin excesiva dependencia sobre otros módulos y abstrayendo los servicios comunes de cada módulo. La prestación de servicios comunes a través de módulos especializados suelen considerarse medidas que apoyan la reutilización. Si los servicios comunes han sido identificados y abstraídos, la modificación de los mismos deberá hacerse sólo una vez, en lugar de en cada uno de los módulos donde los servicios son utilizados. Por otra parte, la modificación de los módulos que utilizan estos servicios no afectará a otros usuarios.

Dichos cambios previstos proporcionan una manera de evaluar la asignación particular de responsabilidades en cada módulo. Para cada cambio previsto, se debe proponer una descomposición del sistema que limite el conjunto de módulos que deben ser modificados para lograr dicho cambio. Así mismo, se deben identificar los cambios que son considerados fundamentales pero diferentes y si éstos afectan a los mismo módulos. Aunque prever cambios o modificaciones futuras no es precisamente ocuparse de la coherencia semántica entre las responsabilidades de los módulos, prever cambios o modificaciones permite ocuparse de la minimización de los efectos sobre los módulos que pudieran ocurrir con cada cambio. Aunque resulta prácticamente imposible anticipar todo el conjunto de cambios que sufrirá el sistema en cuestión.

Así mismo, puede considerarse hacer que los módulos del sistema sean más generales, lo que permitirá calcular o procesar una gama más amplia de funciones basadas en la entrada. Dicha entrada, puede ser considerada como la definición de un lenguaje para cada módulo, el cual puede ser tan simple como hacer parámetros de entrada constantes o tan complicado como implementar el módulo como un intérprete y hacer los parámetros de entrada un programa en el lenguaje del intérprete. Pero la idea de realizar un módulo más general, permite que los cambios solicitados se puedan hacer ajustando el lenguaje de la entrada en lugar de modificar el módulo en si.

Por otra parte, limitar las posibles opciones de modificación sobre los módulos reduce el efecto de dichas modificaciones, es decir, reduce la cantidad de módulos afectados por la solicitud de un cambio o modificación.

La prevención del efecto dominó es una técnica que se ocupa de realizar los cambios a los módulos que no son afectados de manera directa por una modificación. Es decir, después de haber identificado los módulos que serán afectados por los cambios previstos y haber establecido el lenguaje de entrada de cada uno de ellos, es necesario identificar y controlar aquellos módulos que dependen de alguna manera de los módulos que se modificarán. Para ello el sistema debe ser descompuesto en trozos pequeños a manera de que cada trozo ejecute sólo una responsabilidad en específico, es decir, que posea una funcionalidad bien delimitada y definida, para así establecer qué información será de carácter privado y cuál será de carácter público. Estos trozos formaran los módulos del sistema. Aquella información que se considere pública estará disponible a través de interfaces específicas. Lo que permitirá aislar los cambios dentro de un módulo y así evitar la propagación de los cambios. Dichas interfaces deben ser estables y conservar su sintaxis.

Una manera de lograrlo es separar la interfaz de la implementación, lo que permite la creación de interfaces abstractas que ocultan las modificaciones.

El restringir los módulos con los cuales un módulo dado comparte datos, reduce el número de módulos que consumen datos producidos por el módulo dado, es decir, se reduce la relación producción/consumo reduciendo la dependencia entre los módulos y el efecto domino.

En resumen, la propiedad de calidad *Modificabilidad* aporta al sistema las técnicas que facilitaran al usuario agregar sus procedimientos al sistema, sin tener que ocuparse de todos los efectos que sean generados por estos cambios, y brindándole transparencia en la manipulación de los módulos que intervienen en el proceso de adición de dichos procedimientos.

La propiedad de ***Usabilidad*** por su parte, se refiere a la facilidad que le brinda el sistema al usuario para ejecutar una tarea deseada así como el soporte que el sistema le brinda a éste. De hecho, la usabilidad está fuertemente relacionada con la propiedad de Modificabilidad.

La usabilidad envuelve aspectos tanto *arquitectónicos* como *no arquitectónicos*. Los *aspectos no arquitectónicos* incluyen el hacer la interfaz de usuario clara y fácil de usar. Se refiere a la selección de los elementos de diseño que aportarán facilidad de uso y una apariencia limpia y agradable, por ejemplo, elegir entre un radio-button o un check-box. Aunque parecen detalles menores, la correcta selección y ubicación de los elementos en la pantalla, incluso hasta los colores de la interfaz, tienen gran impacto sobre el usuario final e influyen en la aceptación del sistema.

Los *aspectos arquitectónicos* se refieren a si el sistema provee al usuario la habilidad para cancelar operaciones, deshacer operaciones o reusar datos previamente introducidos.

Al igual que en la propiedad de calidad anterior, es recomendable separar la interfaz de usuario del resto del sistema, esto debido a que la interfaz de usuario sólo sirve como el medio que comunica al usuario con los datos que manipula y procesa el sistema, no es propiamente parte integral de los datos manipulados, ni del procesamiento de los mismos además de que permite modificar los aspectos arquitectónicos o no arquitectónicos sin que la modificación de uno de ellos se vea afectado por la modificación del otro.

Durante el diseño del sistema, se pueden capturar las características de otro sistema con el que el usuario se encuentre familiarizado. En caso de ser un usuario completamente nuevo en el ambiente ofrecido por el sistema, el diseñador debe lograr que el usuario aprenda a utilizar el sistema fácilmente.

Una forma en que el usuario se sienta integrado al sistema, es proporcionándole cierta retroalimentación acerca de lo que el sistema esta ejecutando, mediante barras

de estado o ventanas de procesamiento, que indiquen al usuario que su petición esta siendo atendida. Otra forma es proporcionarle la capacidad para emitir comandos que ayuden a ejecutar tareas frecuentes o con las que ya este familiarizado.

Durante la ejecución del sistema, este estará en espera de acciones o eventos que serán generados por el usuario o por el mismo sistema. Para que el sistema pueda dar respuesta a dicho evento, de manera interna y transparente, el sistema debe poseer cierta información (modelo) de lo que sucede. Dicho modelo puede ser referente a una tarea, evento, procesamiento, o referente al estado del propio sistema. Para cada caso, el modelo sirve para determinar qué clase de acción se va a ejecutar y proveer alguna clase de asistencia o brindar una retroalimentación al usuario. Así mismo el modelo permitirá almacenar el estado del sistema, lo que facilitará ofrecer al usuario operaciones de cancelación o de deshacer.

El usuario podrá reutilizar datos previamente introducidos (procedimientos, métodos, clases), esto debido a la separación de la interfaz del resto del sistema, la agrupación de funcionalidades en forma de módulos y clases y la aplicación de la modificabilidad lo que llevará a un sistema que es usable y adaptable a las necesidades del usuario.

En resume la característica más relevante de la propiedad de *usabilidad* que aporta al sistema SPADES son los aspectos arquitectónicos que llevan al diseñador a considerar conocer el estado del sistema antes durante y después de una operación.

2.4. Uso de diagramas para la especificación del sistema

El lenguaje UML (Unified Modeling Language) es utilizado para especificar, analizar, diseñar e implementar sistemas de software. Es un lenguaje de modelado de propósito general estandarizado desarrollado por la OMG (Object Management Group).

La especificación de un sistema mediante los diferentes tipos de diagramas ofrecidos por UML facilita la comunicación dentro de todos los niveles de desarrollo del sistema, ya que uno de sus objetivos es evitar la ambiguedad para que se desarrolle exactamente lo que se especifico.

UML ofrece 14 diferentes diagramas divididos en dos categorías: los *diagramas de estructura* y los *diagramas de comportamiento*. Los diagramas de estructura representan la parte *estática* del sistema mediante el uso de objetos, atributos, operaciones y relaciones. Mientras que los diagramas de comportamiento expresan la colabora-

ción entre los objetos y los cambios en el estado interno de éstos mostrando la parte *dinámica* del sistema.

En esta sección se presentan dos de éste tipo de diagramas: *Casos de Uso* y *Diagramas de Secuencia.*

2.4.1. Casos de uso

El propósito de los diagramas de caso de uso es presentar la funcionalidad del sistema en términos de actores, sus acciones (representadas como casos de uso) y cualquier dependencia entre los casos de uso. Los casos de uso representan qué acciones del sistema son efectuadas por qué elementos o actores.

A continuación se presentan varios diagramas de casos de uso, los cuales muestran varias acciones que debe llevar acabo el usuario para hacer uso del sistema SPADES.

Cuando el usuario capture su sistema requiere adicionar tantos lugares, transiciones y arcos sean necesarios, vea Figura 2.1. Una vez que el sistema este completamente capturado, el usuario puede almacenar su modelo, vea Figura 2.3.

El usuario puede requerir abrir un sistema anteriormente capturado, Figura 2.2. Incluso el usuario puede exportar el sistema capturado en el editor como una imagen, Figura 2.4. SPADES ofrece dos formatos, .png y .ps.

El sistema SPADES permite aplicar procedimientos de análisis a su modelo, Figura 2.5. Así mismo el usuario puede eliminar procedimientos que no requiera o que simplemente no desee se muestren como una opción para ejecutar, Figura 2.6. También el sistema SPADES permite al usuario agregar nuevos procedimientos de análisis, Figura 2.7, característica principal de la herramienta.

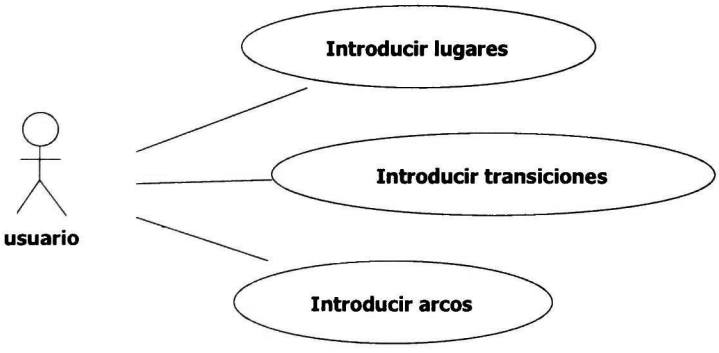


Figura 2.1: Capturar el modelo mediante la red de Perti.

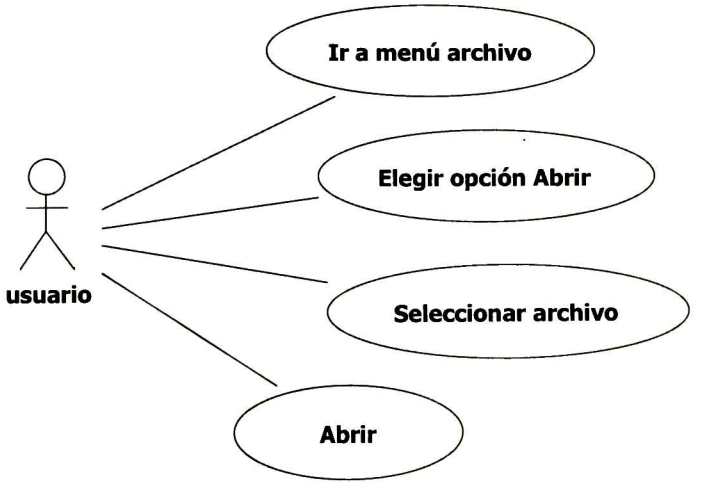


Figura 2.2: Abrir una archivo existente.

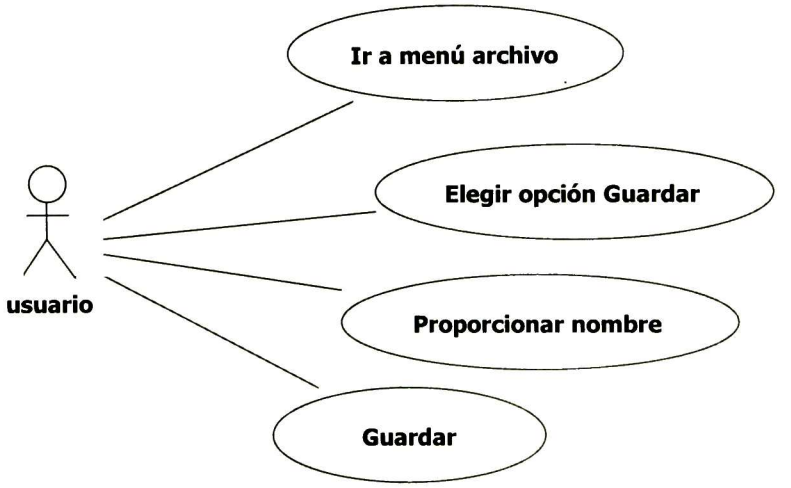


Figura 2.3: Guardar un archivo.

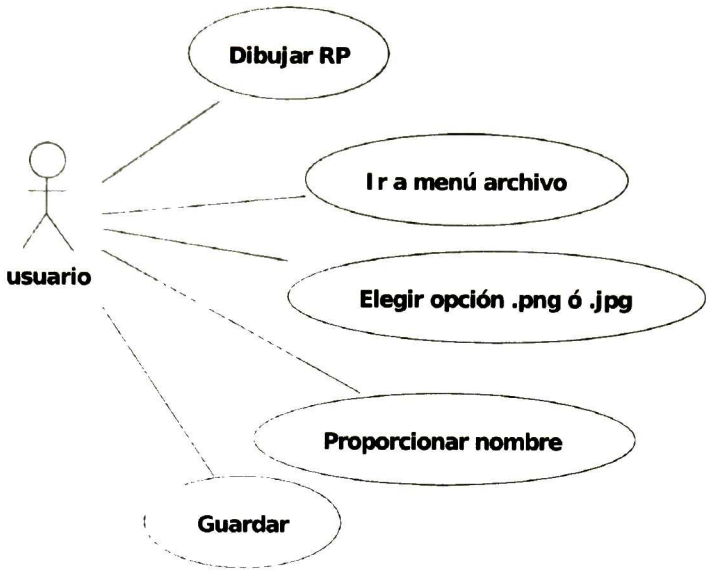


Figura 2.4: Exportar el modelo como una imagen (.png o .ps).

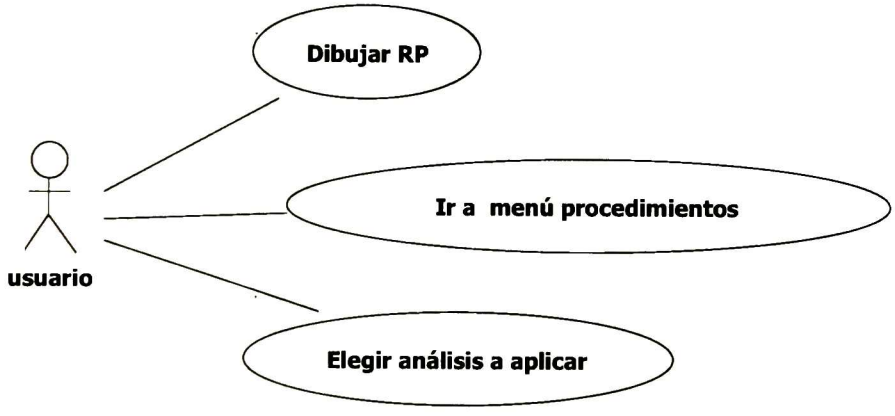


Figura 2.5: Aplicar un procedimiento de análisis al modelo.

Figura 2.6: Eliminar un procedimiento de análisis del sistema SPADES.

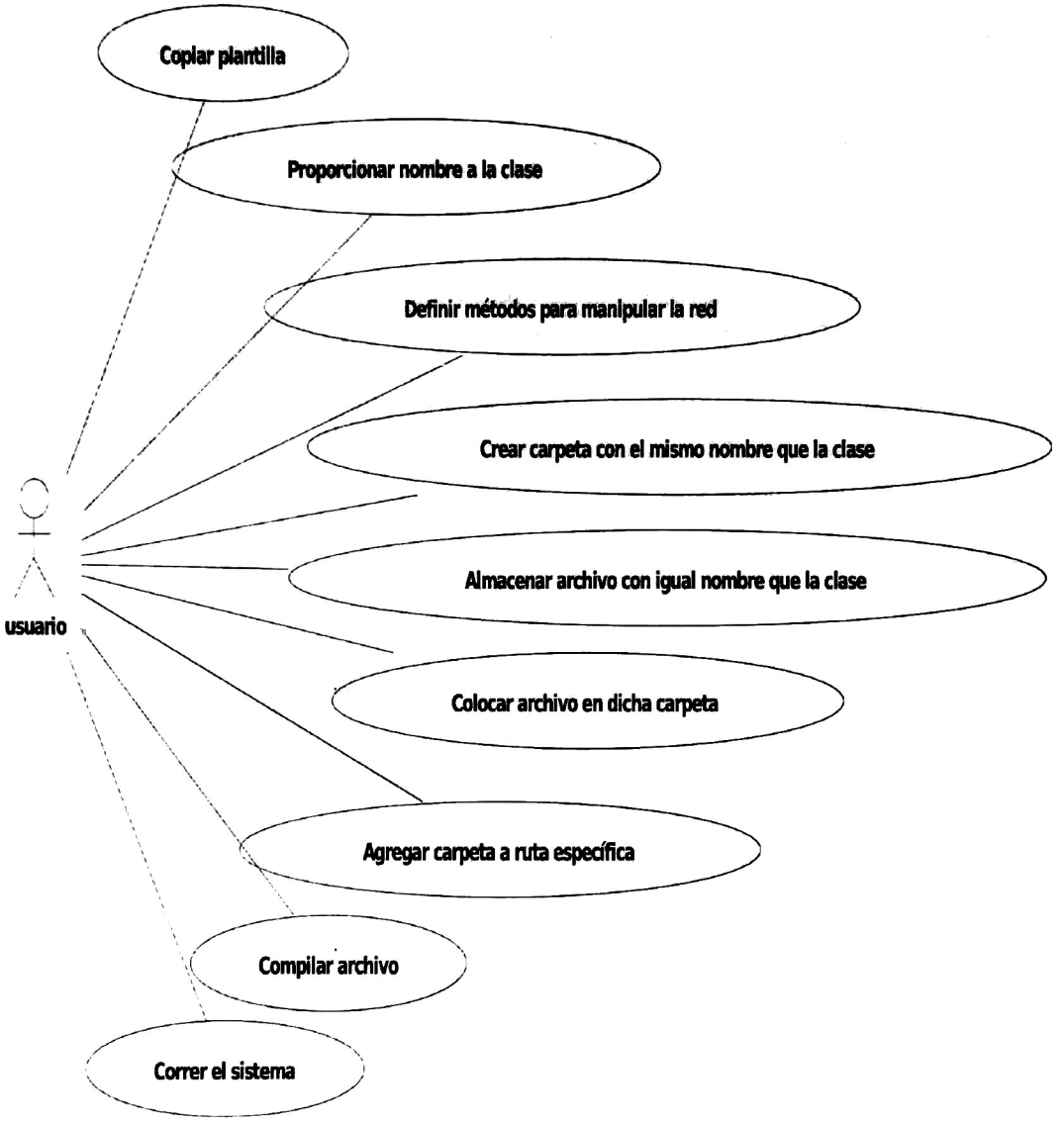


Figura 2.7: Generar un nuevo procedimiento de análisis y agregarlo al sistema SPADES.

2.4.2. Diagramas de secuencia

Los diagramas de secuencia UML modelan la interacción entre objetos de un sistema a través del tiempo, mostrando la secuencia de mensajes entre dichos objetos. A continuación se presentan un conjunto de diagramas bajo distintas acciones iniciadas por el usuario.

Cuando el usuario desea abrir un documento, Figura 2.8, cuando el usuario almacena el sistema capturado en el editor, Figura 2.9.

Otra de las funcionalidades proporcionadas por SPADES, es la de exportar el sistema capturado como imagen, 2.10; la simulación, Figura 2.12; la aplicación de procedimientos de análisis, Figura 2.13; editar las propiedades de arcos, Figura 2.14; o lugares, Figura 2.15. Así como capturar el sistema mediante el dibujo de la red de Petri en el editor, Figura 2.11.

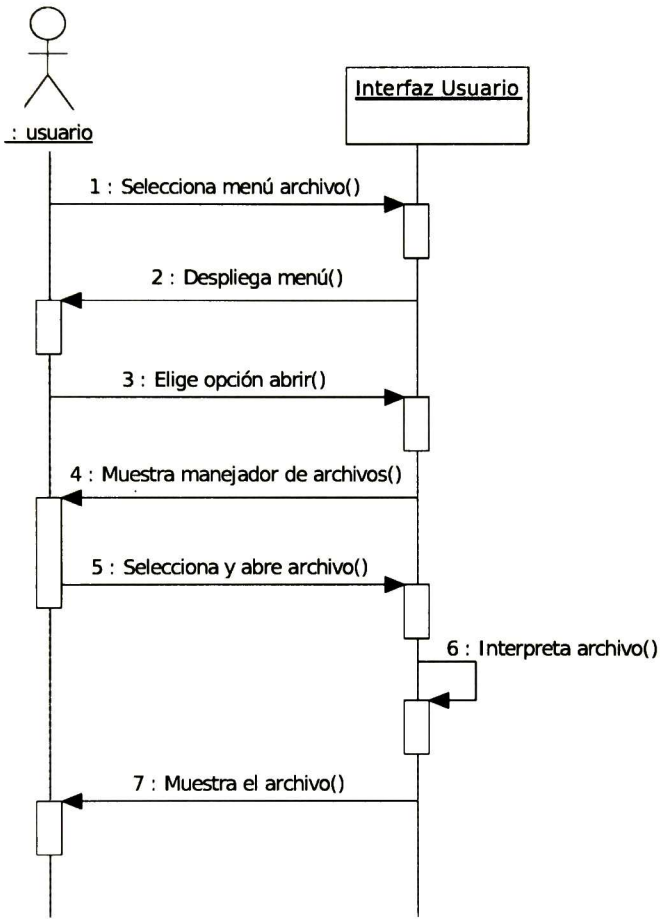


Figura 2.8: Diagrama de secuencia de apertura de un documento existente.

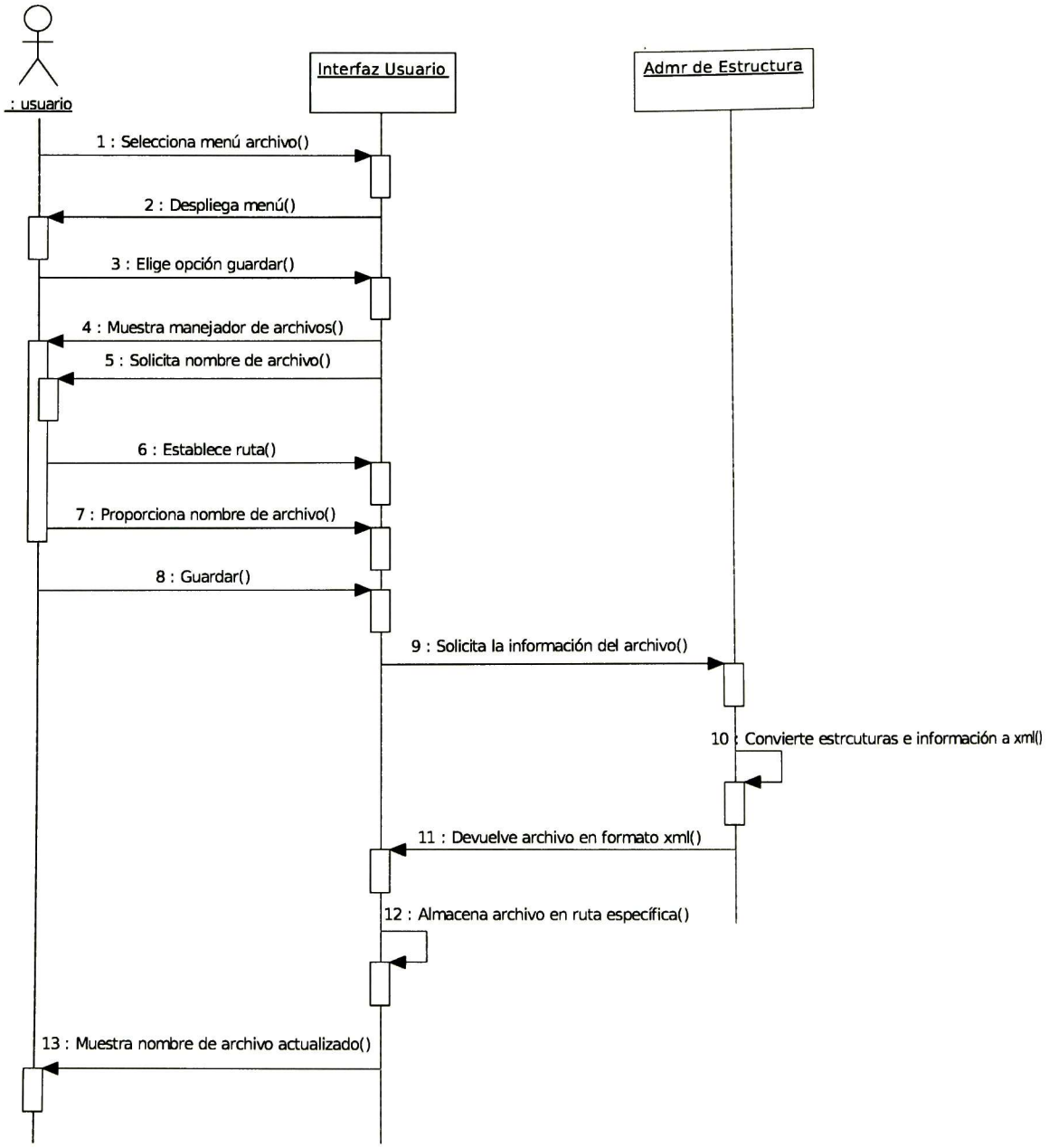


Figura 2.9: Diagrama de secuencia de almacenamiento de un documento.

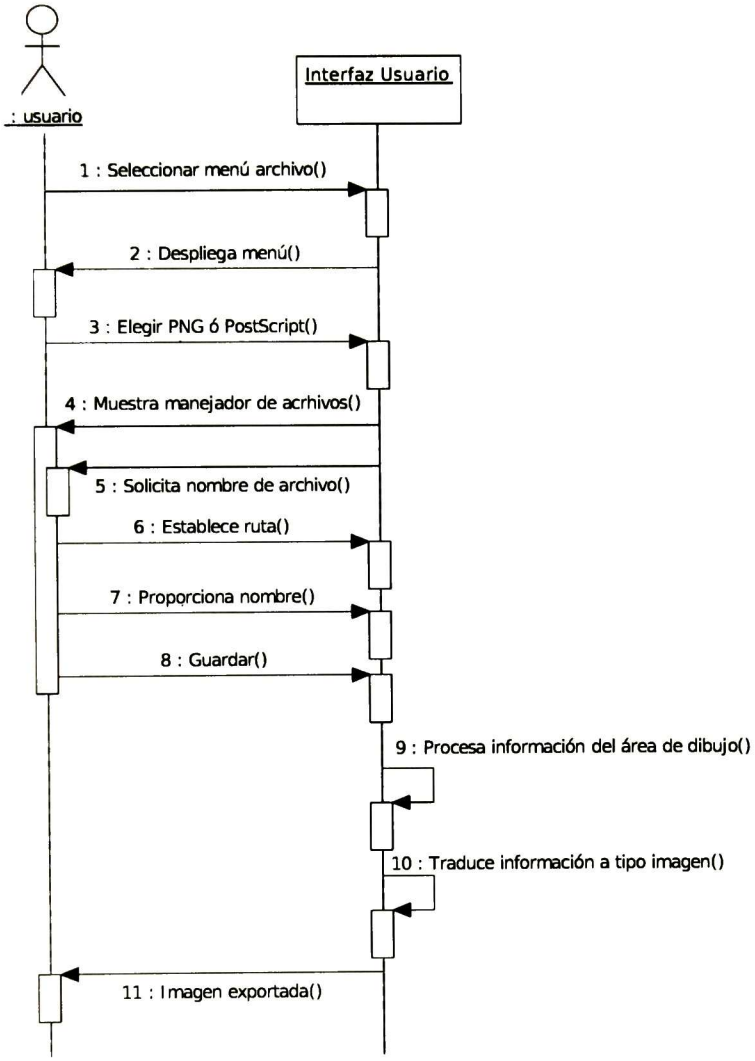


Figura 2.10: Diagrama de secuencia de exportar el modelo como imagen PNG o PostScript.

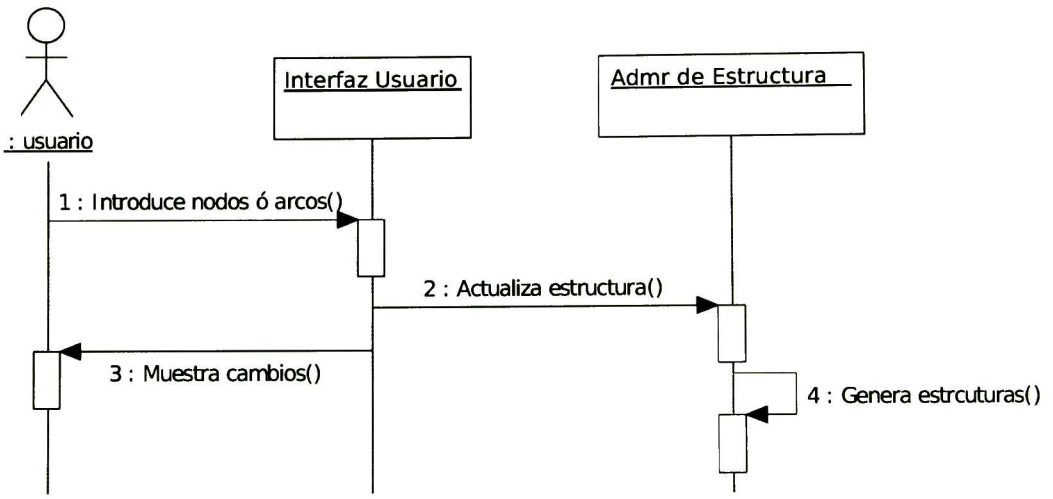


Figura 2.11: Diagrama de secuencia de dibujo de una red de Petri.

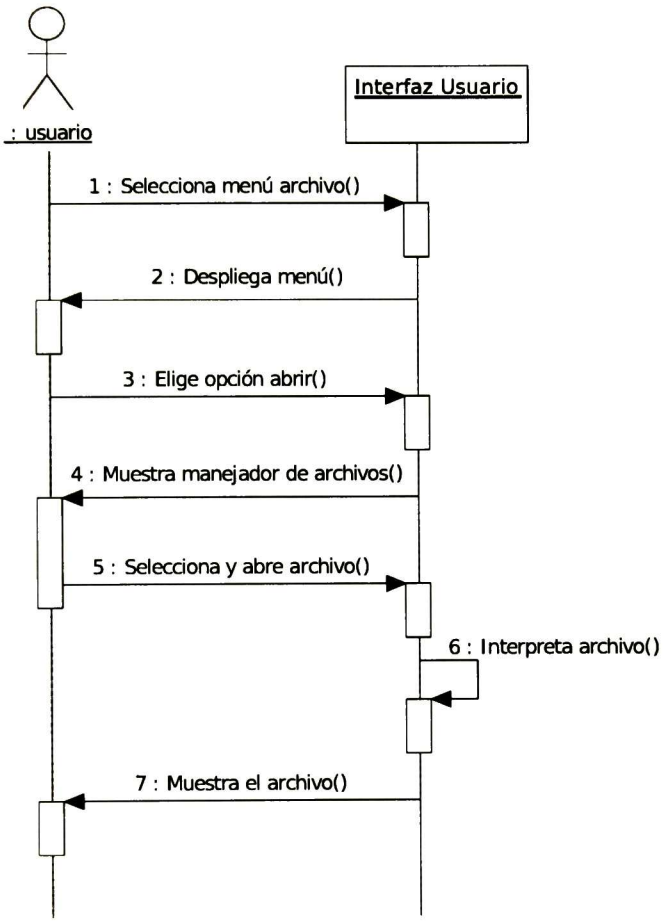


Figura 2.12: Diagrama de secuencia simulación del modelo.

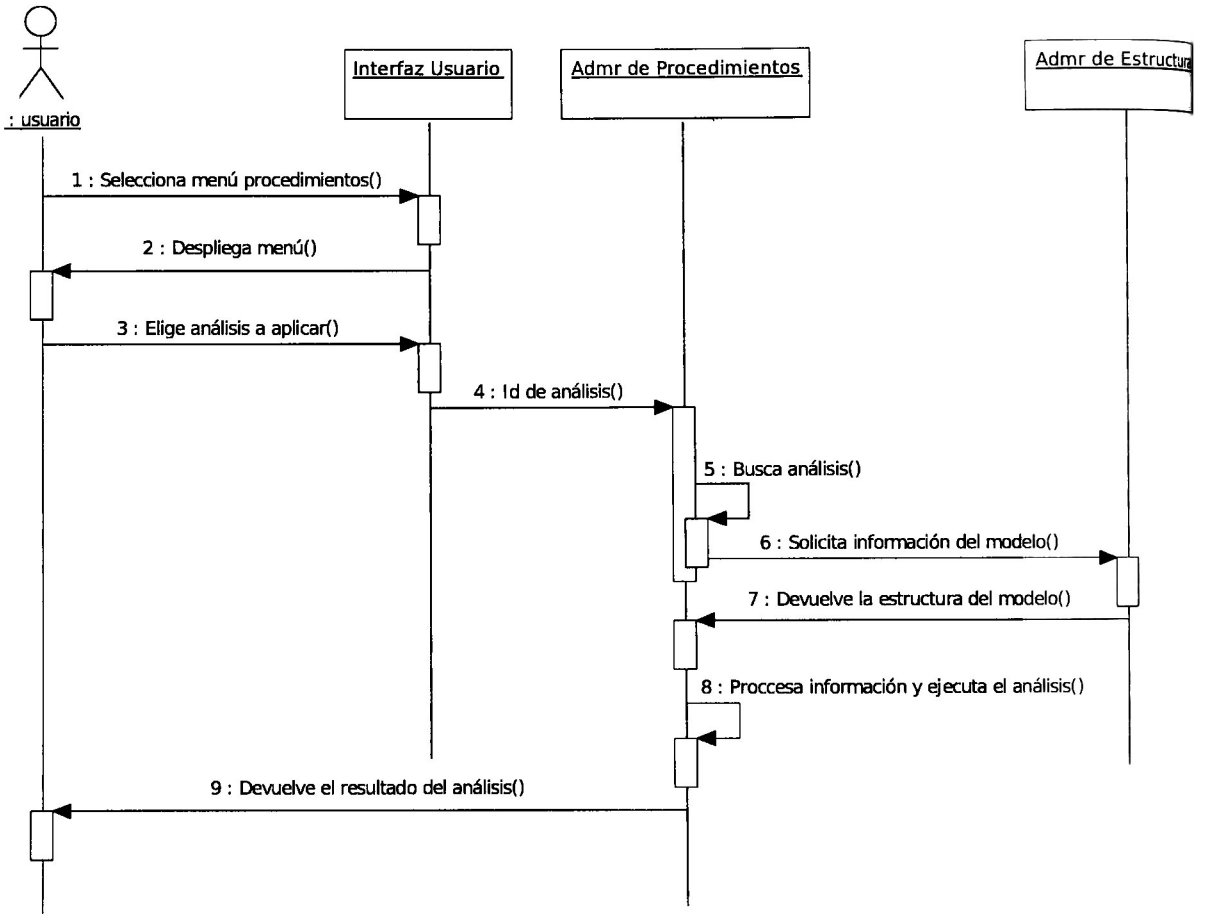


Figura 2.13: Diagrama de secuencia aplicar un procedimiento de análisis.

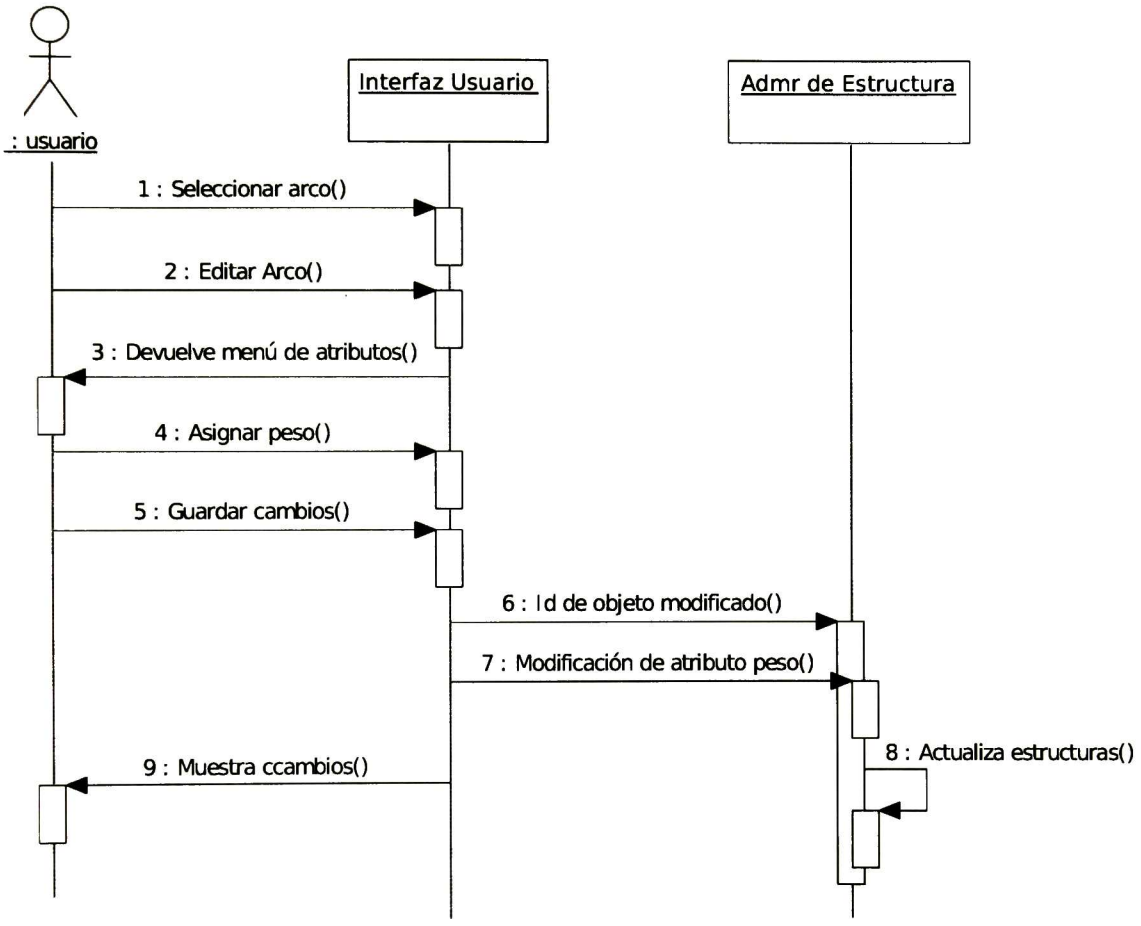


Figura 2.14: Diagrama de secuencia de edición de propiedades de un arco.

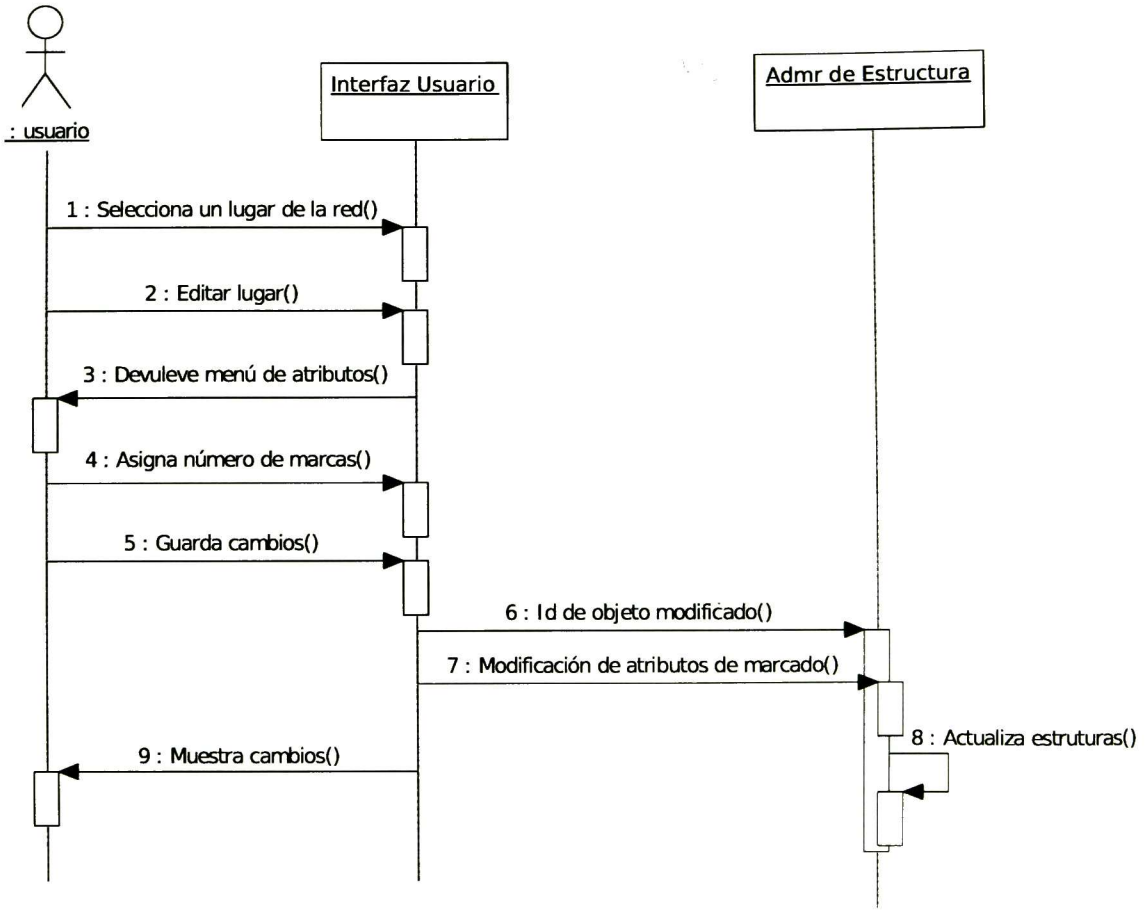


Figura 2.15: Diagrama de secuencia de edición de propiedades de un lugar.

Capítulo 3

Arquitectura de SPADES

En este capítulo se hace una revisión de los patrones arquitectónicos, los cuales ayudan en el diseño de las arquitecturas de software. Además se presentan las decisiones de diseño así como las metodologías que fueron adoptadas para generar una arquitectura que representará toda la funcionalidad requerida. En este capítulo se verá que la arquitectura no surge arbitrariamente, sino que es un proceso de diseño incremental que surge de la expresión de una necesidad y se refina mediante el apoyo en técnicas de diseño ya implementadas.

Aunque las propiedades de calidad para SPADES son consideradas dentro la especificación, éstas propiedades son de mayor impacto en el diseño de la arquitectura, de hecho estas propiedades son denominadas a menudo como *estrategias arquitectónicas*, debido a que tienen un gran impacto en la toma de decisiones para el diseño de ésta.

3.1. Patrones Arquitectónicos

Los patrones de arquitectura son un concepto clave en el campo de la arquitectura de software, estos expresan en un esquema la organización estructural fundamental del sistema, que consiste de subsistemas, sus responsabilidades e interrelaciones. Así mismo provee de la descripción de los elementos y el tipo de relación, junto con un conjunto de restricciones de cómo éstos deben ser utilizados. [3]

Los patrones de arquitectura son conceptos útiles que capturan elementos y propiedades de una arquitectura, más no son propiamente una arquitectura.

Para más detalle sobre los patrones arquitectónicos véase [9], [3].

En [1] se propone un lenguaje de patrones que no es más que un conjunto que agrupa y clasifica los patrones arquitectónicos existentes en forma de ***Vistas***, centrándose

en establecer la relación entre los distintos patrones y presentándolos de la siguiente manera:

Vista Capas

El sistema es visto como una entidad heterogénea compleja que puede ser particionado en partes que interactúan entre si. Estas partes individuales son componentes que se dividen tanto como sea posible una de otra. Los mecanismos de interacción entre estos componentes son implementados mediante conectores que incluyen sus interfaces, estados y protocolos de interacción propios. En esta vista encontramos los patrones arquitectónicos Layers e Indirection Layer.

Vista Flujo de datos

El sistema se visualiza como un número de subsecuentes transformaciones del flujo de datos de entrada. Los componentes que efectúan transformaciones son independientes uno de otro y poseen puertos de de entrada y salida. Los elementos que llevan el flujo de datos son conectores que realizan funciones de lectura y de envío de datos. Los patrones arquitectónicos Batch Sequential y Pipes and Filters se encuentran en esta clasificación.

Vista Centrada en datos

Al sistema se le considera como un almacén persistente y compartido de datos al cual se puede acceder y ser modificado por múltiples componentes.

El almacenamiento de datos y los elementos que pueden acceder son independientes uno de otro. Estos elementos que transfieren, escriben datos o leen del almacén, son los conectores en el sistema. También es posible que exista mas de un almacenamiento de datos. Los patrones arquitectónicos agrupados en esta clasificación son: Shared Repository, Active Repository y Blackboard.

Vista Adaptación

El sistema se conceptualiza como una parte del núcleo que permanece invariable y otra inadaptable que cambia en cualquier momento en el tiempo o en diferentes versiones del sistema. Estos dos elementos (variables y adaptables) son llamados puntos de variación y se comunican uno con otro a través de conectores que poseen interfaces bien definidas. Microkernel, Reflection y Interceptor son patrones arquitectónicos que pertenecen a este tipo de vista.

Vista Interacción de usuario

El sistema es visto como una parte que representa la interfaz de usuario y otra parte que contiene la lógica de la aplicación, asociada con la interfaz de usuario. Ambas partes interactúan una con otra a través de conectores que intercambian mensajes basados en un mecanismo de notificación. Los patrones que aquí se agrupan son:

Model-View-Controller, Presentation-Abstraction-Control y C2.

Vista Distribución

El sistema se conceptualiza como un número de componentes que están físicamente localizados en diferentes nodos de red o se ejecutan en distintos proceso. Los patrones que recaen en esta vista son: Broker, Remote Procedure Calls y Message Queuing.

Se puede observar que existen una amplia variedad de patrones arquitectónicos que pueden ser usados según el tipo de sistema que se desee desarrollar así como en el ambiente en que se desenvolverá y aplicará el sistema. De todos éstos, sólo una vista es la que interesará para el desarrollo de la arquitectura y la cual se especifica más adelante.

3.2. Patrones arquitectónicos de interacción de usuario

Aunque se ha llegado a un consenso en el campo de la arquitectura de software respecto a los patrones arquitectónicos como parte esencial para la descripción de una arquitectura. El describir, buscar y aplicar patrones de arquitectura, en la práctica sigue siendo en gran medida ad-hoc y no sistemático. Sin embargo, los patrones arquitectónicos ofrecen soluciones bien definidas a los problemas arquitectónicos.

De las vistas presentadas anteriormente ***Interacción de usuario*** se centra en dividir el sistema en dos partes: Una de ellas concentra las funcionalidades referentes a la interfaz de usuario y la otra la lógica de la aplicación. Si recordamos, en el capítulo de la *Especificación* ambas propiedades de calidad seleccionadas recomendaban separar el sistema de esta manera. Cada vista poseía distintos patrones arquitectónicos. En este caso, la vista *Interacción de usuario* envuelve tres patrones arquitectónicos: MVC (Model-View-Controler), PAC(Presentation-Abstraction-Control) y C2. De manera particular uno de éstos patrones resulta muy apropiado para el diseño de la arquitectura del sistema SPADES.

A continuación se describen de manera breve cada uno de ellos.

3.2.1. PAC

Presentación-Abstracción-Control (Presentation-Abstraction-Control) es usado como una estructura para sistemas de software interactivos que se implementa como un árbol de agentes jerárquico cooperativos. Cada uno de los agentes consiste de una

triada formada por presentación, abstracción y control, y es responsable de un aspecto específico de la funcionalidad de la aplicación. Los agentes en sus triadas se comunican con otro agente a través de la parte de control de cada triada aislando la presentación y la abstracción. En cada agente, los componentes de presentación y abstracción nunca se comunican uno con otro, de hecho el componente de presentación es en esencia sólo un filtro que toma los datos en bruto que el controlador le provee mostrándolos como HTML o WML o XML o texto, etc. El componente de presentación puede considerarse como un sistema de plantillas que muestra el comportamiento visible de los agentes. El componente de abstracción por su parte mantiene el modelo de datos subyacente de cada agente y provee la funcionalidad para manipularlos. [6]

Aunque PAC aísla funcionalidades en tres componentes distintos: Presentación, Abstracción y Control, es decir, la interfaz, el modelo de datos subyacente en cada agente y la comunicación respectivamente. Los agentes jerárquicos son utilizados en sistemas dinámicos que generalmente ofrecen diversas funcionalidades las cuales necesitan ser presentadas al usuario a través de una interfaz coherente, esta variedad de funcionalidades puede requerir su propia interfaz a la vez que es necesario comunicarse con otras funcionalidades para lograr una meta mayor. Característica que no se apega a las necesidades del sistema SPADES, ya que no se desenvuelve en ambientes de este tipo. Aunque cabe mencionar que la división de funcionalidades apoyaría la propiedad deseable de la modificabilidad, por su parte, mediante el conocimiento del modelo de datos de cada agente, la usabilidad. Pero se desaprovecharía la funcionalidad proporcionada por los agentes.

3.2.2. C2

El sistema es descompuesto en una jerarquía top-to-bottom de componentes concurrentes que interactúan asíncronamente enviando mensajes a través de los componentes que actúan como conectores. Los componentes solicitan mensajes de petición hacia arriba de la jerarquía, con conocimiento de los componentes superiores, y de igual manera los componentes envían mensajes de notificación hacia abajo en la jerarquía, pero sin conocimiento de los componentes que se encuentran por debajo de ellos. Los componentes sólo se encuentran conectados mediante conectores, pero los conectores pueden estar conectados con ambos (componentes y conectores). El propósito de dichos conectores en este patrón arquitectónico es hacer broadcast, rutear y filtrar mensajes [1].

Este patrón arquitectónico se orienta a sistemas que se desenvuelven en ambientes

de red, en los que se puede requerir la implementación de distintos tipos de lenguajes para la manipulación de los distintos componentes, diferentes estructuras de datos y lógica de aplicación en las distintas capas que forman los conectores. Así como una interfaz que permita la visualización o manipulación de los cada componente. Por lo que C2 no es adecuado para el sistema SPADES.

3.2.3. MVC

En el patrón arquitectónico MVC el sistema es dividido en tres diferentes partes: el *modelo* que encapsula algunos los datos de la aplicación y la lógica que manipulará dichos datos, independientemente de la interfaz de usuario; una o múltiples *vistas* que mostrarán una porción en específico de los datos a el usuario; un *controlador* asociado con cada vista el cual recibirá entradas del usuario y las traducirá a una solicitud al modelo. Las vistas y los controladores constituyen la interfaz de usuario. El usuario interactuará estrictamente a través de las vistas y sus controladores, independientemente del modelo, el cual, en su momento notificará a todas las interfaces de usuario acerca de las actualizaciones que se requieran realizar en el modelo, debido a las modificaciones hechas por el usuario. [1]

En base a los requerimientos no funcionales (descritos en el capítulo 2), el patrón arquitectónico MVC, se adapta a tales necesidades. De hecho, éste patrón arquitectónico permite realizar la separación de la interfaz del resto del sistema (vista) y además sugiere la separación de la lógica de la aplicación que manipulará los datos del sistema (modelo) de la lógica de control que se encargará de la administración de las entradas y eventos que se generen en el sistema (controlador). Lo cual ayuda al cumplimiento de las propiedades de calidad.

Aunque MVC fue creado en los años 1978-1979, actualmente es un patrón arquitectónico ampliamente utilizado. Este patrón separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos:

Modelo

Es la representación del dominio específico de la información (los datos) sobre en la que la aplicación opera. El dominio lógico de los datos asegura la integridad de estos y permite derivar otros nuevos.

El modelo representa conocimiento. Un modelo podría ser un único objeto o podría ser algún tipo de estructura de objetos. Debe haber una relación uno a uno entre el modelo y sus partes, y el mundo representado como lo percibe el propietario del modelo sobre el otro. [16], [17]

Vista

Convierte el modelo en un formato apropiado para la interacción, es una representación visual de dicho modelo. Típicamente un elemento de la interfaz de usuario.

Una vista esta ligada a su modelo (o a una parte de él), es decir, actúa como un filtro de presentación, y obtiene los datos necesarios a visualizar de su modelo, mediante solicitudes. Incluso puede actualizar el modelo mediante mensajes apropiados. Dichas solicitudes y mensajes deben realizarse bajo la terminología del modelo, por lo que la vista debe conocer la semántica de los atributos del modelo al que representa. [16], [17]

Controlador

El controlador es el elemento que actúa como un intermediario entre el usuario y el sistema. Procesa y responde a eventos, generalmente a acciones de usuario. El controlador puede invocar cambios en el modelo y probablemente en la vista del mismo. [16], [17], [18].

El controlador recibe las peticiones de usuario, las traduce en el mensaje apropiado y las pasa a una o más vistas.

Muchos sistemas informáticos utilizan un sistema de gestión de base de datos para tratar los datos. En le patrón arquitectónico MVC ésta tarea corresponde al modelo. Aunque el MVC puede tener diferentes implementaciones, el flujo de control es generalmente como sigue:

1.- El usuario interactúa con la interfaz de usuario de alguna manera (por ejemplo, presionando un botón).

2.- El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega frecuentemente a través de un gestor de eventos (handler) o callback.

3.- El controlador accede al modelo y lo actualiza de acuerdo a la acción solicitada por el usuario, resultando en un posible cambio en el estado del modelo.

4.- El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene los datos del modelo para generar la interfaz apropiada donde se reflejen los cambios en el modelo.

5.- La interfaz de usuario espera por más interacciones de usuario, comenzando el ciclo nuevamente.

En la figura 3.1 se presenta de manera gráfica dicha interacción entre los componentes del MVC.

En [9], [16], [17] puede encontrase más información acerca de éste patrón arquitectónico.

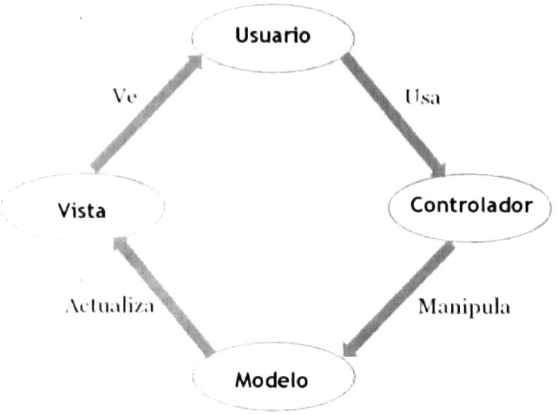


Figura 3.1: Flujo de interacción entre el modelo, la vista y el controlador.

3.3. Diseño de la arquitectura

El diseño de la arquitectura integra la implementación de un patrón arquitectónico y la metodología ADD (Attribute-Driven Design) la cual se basa en el proceso de descomposición recursivo donde, en cada etapa se hace una descomposición de los atributos de calidad que el software tiene que cumplir junto con los requerimientos funcionales para generar módulos que expresen una funcionalidad en específico, tratando siempre de diseñar una arquitectura que satisfaga ambos requerimientos.

La metodología de ADD establece como primer paso identificar los ***architectural drivers***, es decir, los objetivos de máxima prioridad para el sistema y que se consideran tendrán mayor impacto sobre la arquitectura.

Para este caso, los architectural drivers de SPADES son:

- El sistema debe permitir la adición de nuevos procedimientos de análisis sin que sea necesario que el usuario conozca toda la estructura del sistema.
- El usuario debe poder acceder a la definición de la estructura de la red de Petri.
- El sistema SPADES debe ofrecer un editor mediante el cual es usuario plasme su modelo como una red de Petri.
- La eliminación de cualquiera de lo procedimientos no debe afectar la estructura general del sistema.
- El sistema debe permitir la simulación de los modelos plasmados en el editor.

- El lenguaje a utilizar para el desarrollo de SPADES debe permitir generar una herramienta que se ejecute en cualquier computadora independientemente del sistema operativo que ésta tenga.

  El sistema debe cumplir con la propiedad de la modificabilidad y la usabilidad.

La metodología ADD recomienda iniciar con todo el sistema, para después ser descompuesto en subsistemas que después serán particionados en submódulos. Los *architectural drivers* anteriormente descritos, definen los objetivos de prioridad del sistema, los cuales durante el proceso iterativo de descomposición se convertirán en la arquitectura del sistema.

En el Capítulo 2 observamos que las *propiedades de calidad* recomendaban separar la interfaz de usuario del resto del sistema. De igual manera el *patrón arquitectónico MVC* define al sistema como un conjunto de tres partes, en las que una de ellas es la interfaz de usuario. Por consiguiente, el primer paso es separar la interfaz del resto del sistema, dando origen a la primera opción de arquitectura. En la Figura 3.2 puede apreciarse esta primera vista de la arquitectura.

El resto del sistema engloba las funcionalidades de la simulación de la red, la adición de los nuevos procedimientos, así como la definición de la estructura de la red de Petri. La propia *interfaz de usuario* tendrá a su cargo el manejo de todos lo eventos generados por el usuario, así como las distintas ventanas o menús. Vea Figura 3.3.

Aun cuando la *interfaz de usuario* tiene a su cargo el editor gráfico, el conjunto de menús, las barras de herramientas y las diferentes ventanas, la interfaz de usuario sigue expresando la misma funcionalidad. Es decir, tiene a su cargo el control y manejo de todos los eventos generados en los *frames* de la interfaz. Sin embargo, *el resto del sistema* expresa más de una funcionalidad, *Simulador de modelos, Administrador de procedimientos, Administrador de estructura de la red de Petri*, por lo que cada una de éstas funcionalidades deben ser un subsistema por si mismas. Con lo cual obtenemos una arquitectura como la mostrada en la Figura 3.4. Lo que resta es descomponer estos nuevos subsistemas para averiguar si se originaran ortos nuevos, de lo contrario sólo se debe modularizar la funcionalidad de cada uno de ellos.

En la Figura 3.5 se puede apreciar los cuatro subsistemas que forman la arquitectura, *Interfaz de usuario*, *Administrador de estructura de RP*, *Simulador de modelos* y *Administrador de procedimientos*. En esta etapa, los subsistemas expresan una sola funcionalidad. El *Simulador de modelos* se encarga del manejo del algoritmo de simulación y de los métodos que permiten obtener el estado actual en el que se encuentra

la red de Petri capturada por el usuario, así como de los métodos que bloquean los menús o barras de herramientas que puedan alterar la consistencia de la red de Petri.

El *Administrador de la estructura de la red de Petri* contiene las clases que definen a una red de Petri, así como un conjunto amplio de métodos que permiten obtener información de los lugares, transiciones y arcos. En éste subsistema, también se definen métodos que permiten obtener el marcado de la red y su matriz de incidencia. El administrador de estructura gestiona el etiquetado e identificación de los elementos de la red de Petri.

El *Administrador de procedimientos* se encarga de proporcionar la funcionalidad del *Java reflection Api.* La cual, como se explica más adelante, es una característica de Java fundamental para la adición de los nuevos procedimientos de análisis. En éste subsistema se encuentran las clases y métodos que le permiten al usuario visualizar los resultados que se obtienen de aplicar un algoritmo de análisis, así como de ejecutarlos y obtener la definición actual del sistema capturado por el usuario.

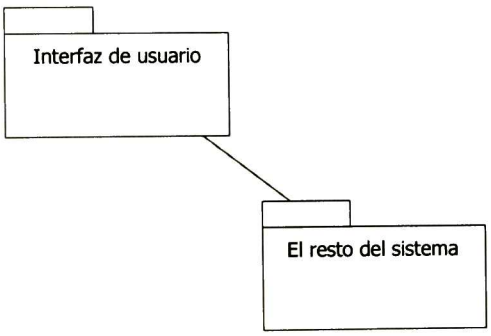


Figura 3.2: La arquitectura de SPADES concebida en dos subsistemas, *la interfaz de usuario* y el resto de la funcionalidad del sistema.

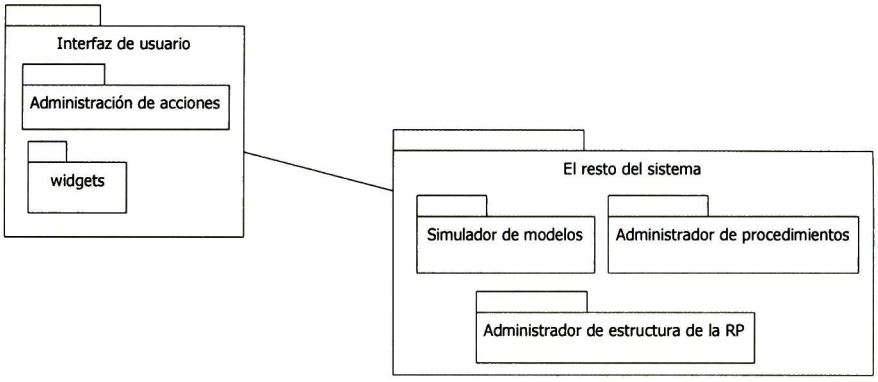


Figura 3.3: La *interfaz de usuario* y *el resto del sistema* y las funcionalidades de las que están a cargo.

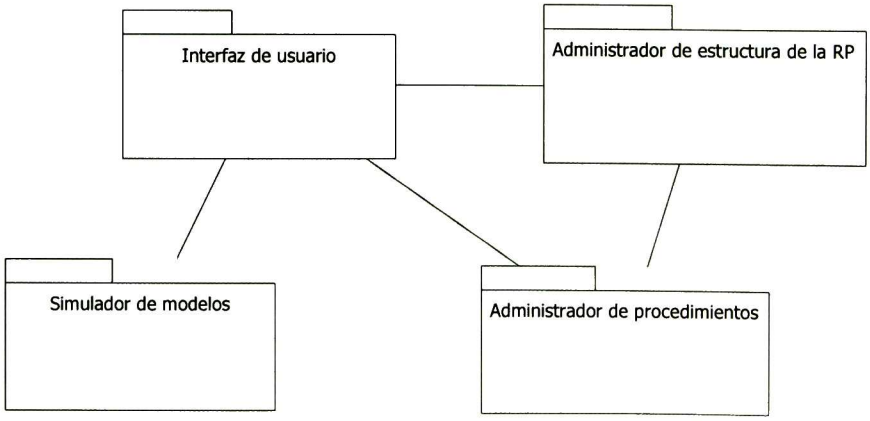


Figura 3.4: La *interfaz de usuario* modularizada, *el resto del sistema* dio origen a nuevos subsistemas.

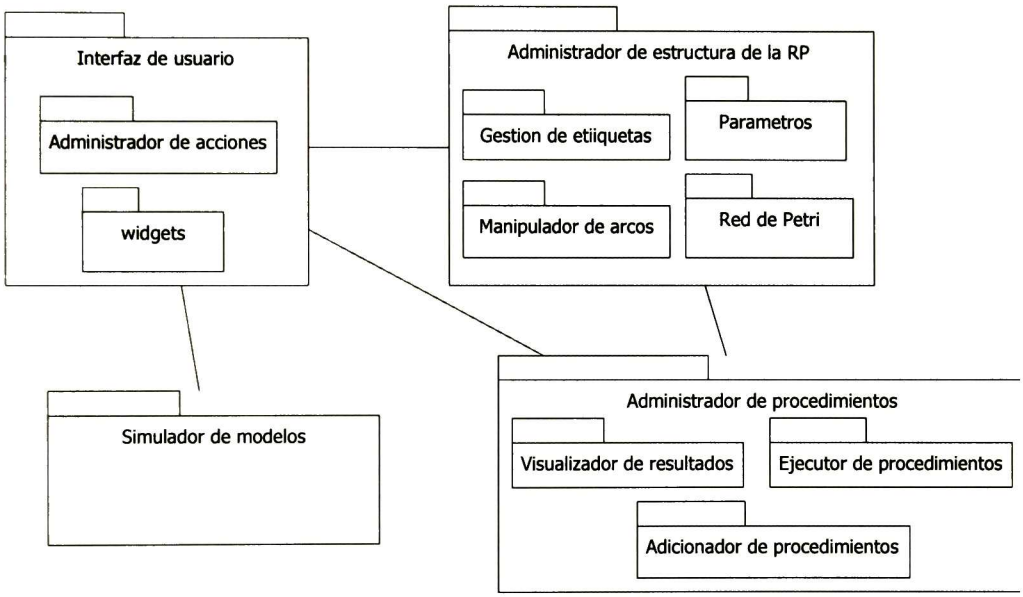


Figura 3.5: La arquitectura del sistema SPADES.

Capítulo 4

Funcionalidades Provistas por el Sistema SPADES

En este capítulo se describen las funciones ofrecidas por el sistema SPADES, las cuales incluyen captura del sistema mediante redes de Petri, la simulación del sistema, y el análisis. Además de las funciones de manipulación de archivo y exportación del modelo como imagen.

De igual manera se describe la forma en que el usuario puede agregar y eliminar procedimientos de análisis. función que permite al usuario adaptar el sistema SPADES a sus necesidades y a los nuevos procedimientos que se generen, principal característica del sistema SPADES.

En este capítulo también se describe como el usuario puede adicionar nuevas extensiones de redes de Petri partiendo de las clases base que le provee el sistema SPADES.

Las palabras que se muestran en letras *cursivas*, hacen referencia a la funcionalidad que se esté describiendo, ya sea el nombre de un menú o alguna de sus funciones, cuando se describen las barras de herramientas, etc.

4.1. Presentación de los Menús

La herramienta SPADES contiene cinco menús: *File*, *Edit*, *View*, *Simulation* y *Analysis Procedures*. Los cuales ofrecen distintas opciones que permiten manipular el modelo.

4.1.1. File

Este menú ofrece opciones de manipulación de archivo como son: *New* que permite generar un nuevo documento y es presentado en un nuevo tab en el editor de SPADES. *Open* permite abrir un documento existente, mostrando una ventana en la que permite al usuario hacer la búsqueda del documento. En la figura 4.1 se puede apreciar el contenido del menú *File.*

Save permite guardar en disco un documento y presenta una ventana en la que el usuario puede nombrarlo como desee y escoger la unidad o ubicación de su almacenamiento, por su parte *Save as* permite renombrar un documento abierto y existente o en el caso de ser la primera ves que se va almacenar, funciona de la misma manera que *Save.* Figura 4.2. La extensión de los documentos de SPADES es .xml.

Close permite cerrar un documento abierto. Si el documento tiene modificaciones sin almacenar, se le pregunta al usuario si desea almacenar los cambios realizados. Si nunca ha sido almacenado se comporta como *Save*, solo que al finalizar el almacenamiento el documento se cierra.

El sistema SPADES ofrece opciones de exportación del modelo como imagen en formatos no susceptibles al degradado de la imagen como son *PNG* y *PostScript.* Cuando se exporta la imagen SPADES muestra una ventana en la que el usuario puede introducir un nombre para la imagen y almacenarla en la ubicación que desee. Figura 4.3 y 4.4.

El usuario también puede imprimir su modelo mediante la opción *Print.* Se ofrece un dialogo de impresión en el que el usuario puede seleccionar la impresora, el número de copias, entre otras distintas propiedades de impresión ofrecidas por la impresora y que varían de acuerdo a la misma. Figura 4.5.

Por último la opción *Exit* la cual cierra el sistema SPADES. Si existe un documento del cual no se hayan almacenado los cambios efectuados, el sistema ofrecerá la opción para realizar el almacenamiento de los mismos.

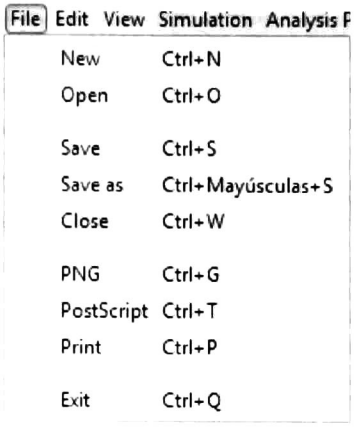


Figura 4.1: Menú que presenta las opciones de manipulación de archivo.

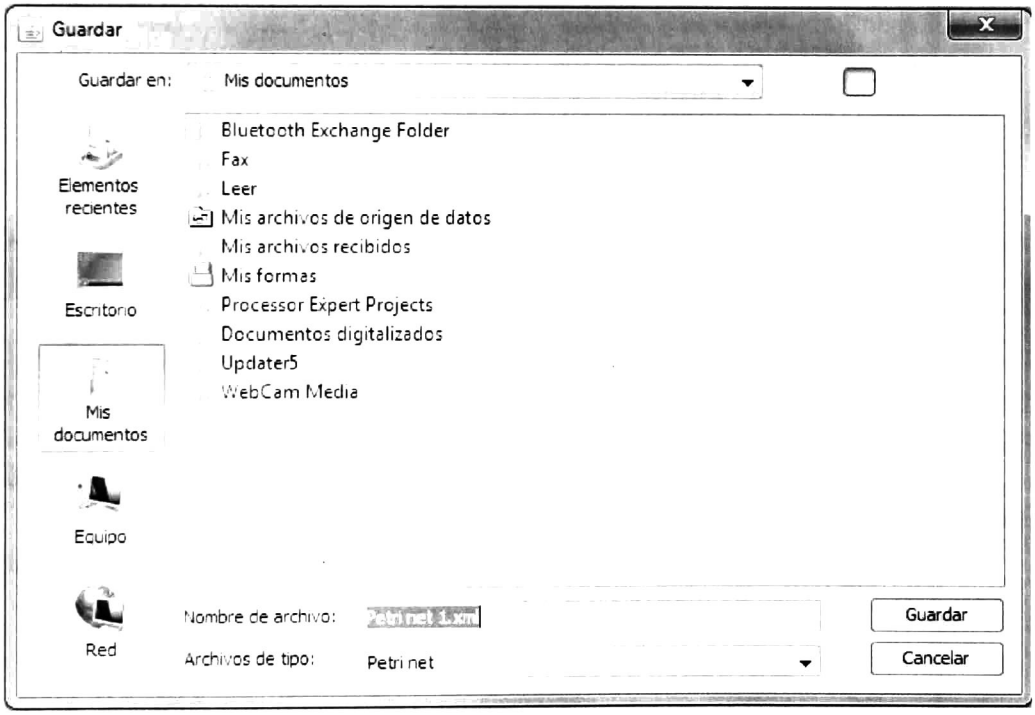


Figura 4.2: Ventana que se muestra cuando el usuario desea guardar un documento.

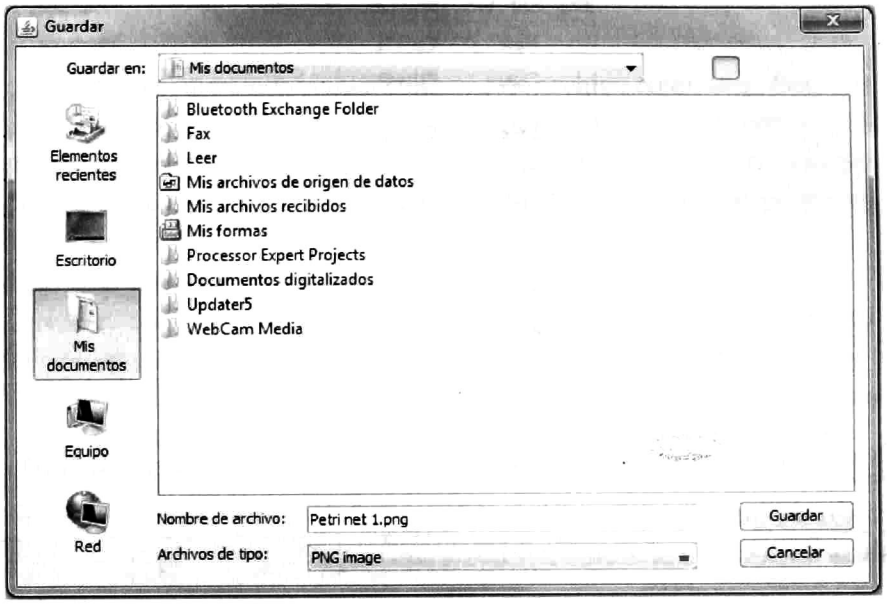


Figura 4.3: Ventana de exportar imagen como PNG.

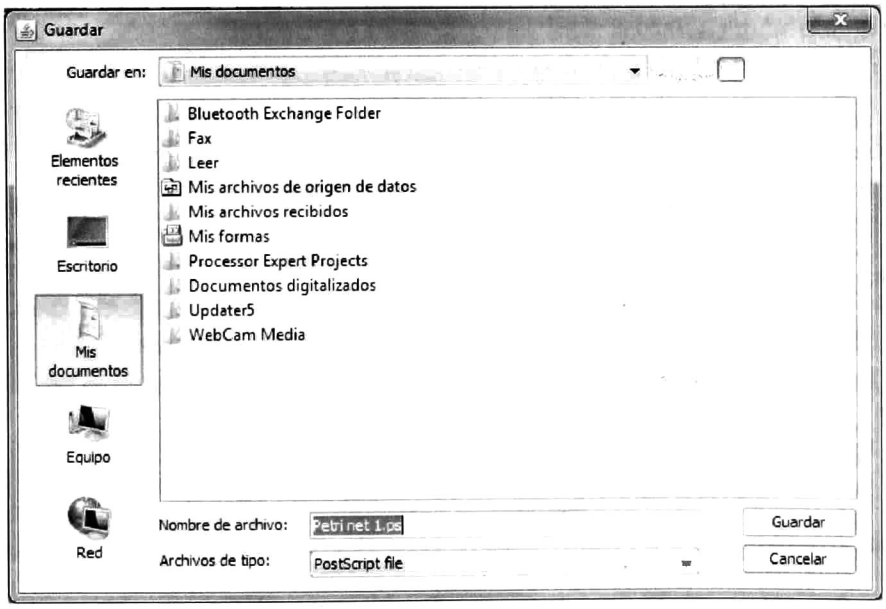


Figura 4.4: Ventana de exportar imagen como PostScript.

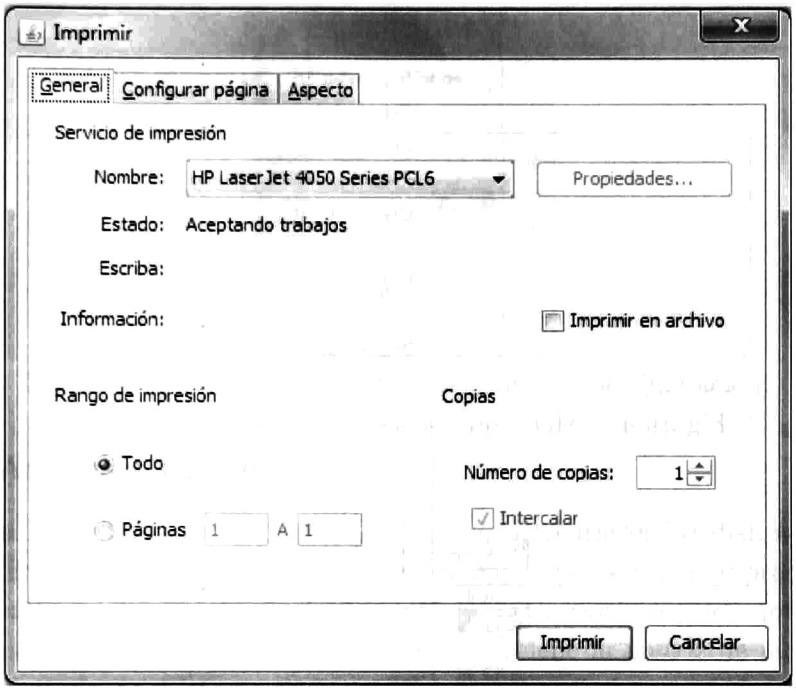


Figura 4.5: Dialogo de impresión de documento.

4.1.2. Edit

Este menú ofrece opciones que permiten manipular elementos de la red que se modela (vea figura 4.6). *Copy* permite copiar elementos individuales o toda la red, *Cut* los corta del editor, mientras que la opción *Paste* pega los elementos que hayan sido copiados o cortados en el editor.

Las opciones *Undo* y *Redo* permiten deshacer y rehacer (respectivamente) la ultima acción de edición llevada acabo por el usuario sobre el modelo.

Delete Borra del editor el o los elementos seleccionados por el usuario.

4.1.3. View

Este menú permite hacer acercamientos *Zoom in* así como hacer *zoom out* al modelo plasmado por el usuario. Aplicar estas funciones no afecta el modelo para su impresión o exportación como imagen. Vea figura 4.7.

Figura 4.6: Menú que presenta las opciones de edición.

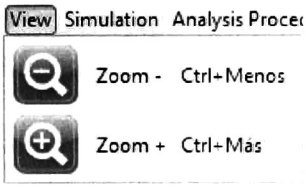


Figura 4.7: El menú view permite hacer zoom in y zoom out.

4.1.4. Simulation

Este menú presenta las opciones de control de la simulación (vea figura 4.8). *Start Simulation* da inicio al modo de simulación manual, en la cual el usuario necesita realizar el disparo de las transiciones habilitadas. La barra de herramientas de dibujo desaparece del editor para dar lugar a la barra de simulación. Cuando se inicia la simulación el modo de simulación automática se habilita.

Back y *Forward* permiten ir un disparo atrás o uno adelante (respectivamente), de las transiciones ya disparadas durante la simulación.

Automatic Simulation ejecuta los disparos de las transiciones habilitadas de manera automática. Sin la necesidad de la intervención del usuario.

Figura 4.8: Menú que presenta los controles de simulación.

4.1.5. Analysis Procedures

Analysis Procedures es el menú en el que se presentarán los distintos procedimientos de análisis (vea figura 4.9). De manera inicial posee sólo tres procedimientos *InvariantAnalysis*, el cual permite obtener los *p* y *t invariantes* del modelo, y *MatrixIncidence* y *Matrices* mediante los cuales se obtiene la matriz de incidencia del modelo presentada de diferente manera. Este menú crecerá en tamaño dependiendo del usuario, debido a que es en éste menú en el que aparecerán los nuevos procedimientos que se adicionen. La manera en que el usuario puede adicionarlos se describe en la sección 4.5.

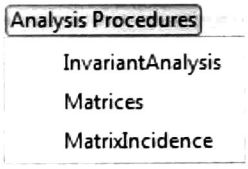


Figura 4.9: Menú que contiene los procedimientos de análisis.

4.2. Edición Lugares, Transiciones y Arcos

Los lugares, transiciones y arcos que forman el modelo plasmado por el usuario, pueden editarse. Para editarlos se requiere hacer uso de la herramienta de selección, seleccionar el elemento que se desea editar, hacer un click con el botón secundario del ratón y parecerá un menú, el cual varía de acuerdo al elemento seleccionado.

Cuando se desea editar un lugar aparece un menú como el mostrado en la figura 4.10 , al seleccionar *Edit Place* se muestra una pequeña ventana como la mostrada en la figura 4.11 Como puede observarse puede editarse la etiqueta mostrada y el número de marcas. La opción *Delete* borra el lugar.

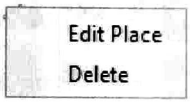


Figura 4.10: Menú de edición de un lugar.

Figura 4.11: Ventana de edición de propiedades de un lugar.

Si lo que se seleccionó para editarse es una transición el menú que se muestra es como el que muestra la figura 4.12. La opción de *Delete* borra la transición. *Edit Transitión* muestra un ventana que permite modificar la etiqueta de la transición y el ángulo de inclinación de la misma (vea figura 4.13).

Figura 4.12: Menú de edición de una transición.

Por su parte el arco muestra un menú (vea figura 4.14) que ofrece las opciones: *Edit Weight* el cual muestra una ventana que permite introducir el peso del arco (vea figura 4.15). Por default este valor siempre es uno. *Insert Point* inserta un punto en el arco permitiendo modificar su ángulo. Pueden agregarse tantos puntos como se desee. La opción de *Delete* funciona igual que en el caso de los lugares y las transiciones.

Figura 4.13: Ventana de edición de propiedades de una transición.

Figura 4.14: Menú de edición de un arco.

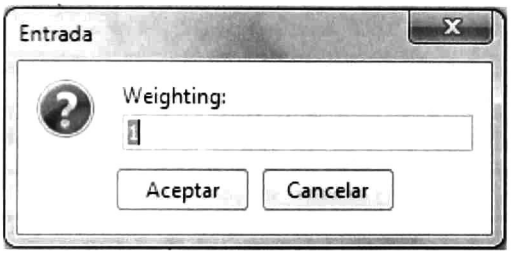


Figura 4.15: Ventana de edición del peso de un arco.

4.3. Captura del sistema mediante redes de Petri

La captura del sistema es una funcionalidad básica que debía ser provista por la herramienta, debido a que es mediante ello que el usuario introduce los nodos y arcos necesarios para formar la red de Petri que representa el sistema.

Para que el usuario pueda capturar su modelo, requiere hacer uso de las herramientas de dibujo. Ahí encontrará la herramienta de selección, lugares, transiciones y arcos, mediante los cuales introducirá y manipulará los elementos de la red de Petri. Figura 4.16.

La editor gráfico de la herramienta fue diseñado para ser funcional, amigable y fácil de utilizar. Los elementos de la red fácilmente pueden ser adicionados y manipulados

en el área de dibujo. Figura 4.17.

Así mismo las transiciones, lugares y arcos pueden ser editados de manera individual. SPADES provee ciertas características de edición que permiten modificar el ángulo, número de marcas y peso respectivamente, así como la etiqueta de éstos. Figuras 4.15, 4.11 y 4.13.

Figura 4.16: Barra de herramientas de dibujo (selección, lugar, transición y arco de izquierda a derecha).

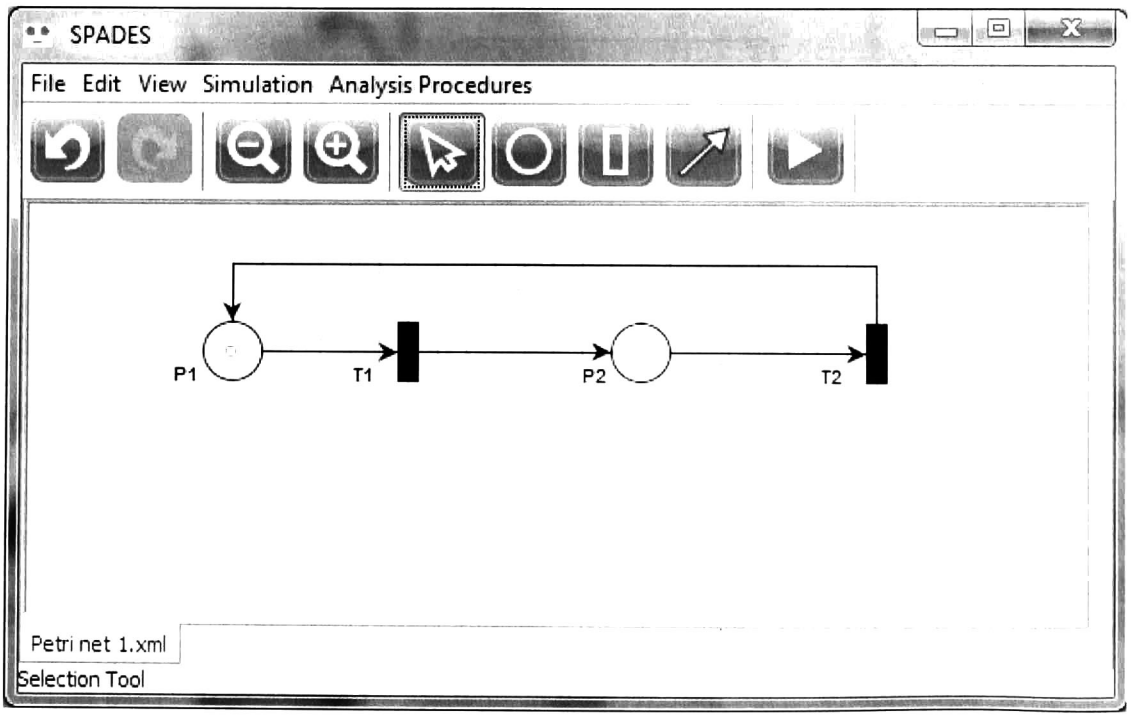


Figura 4.17: Vista del sistema SPADES.

4.4. Simulación del modelo

La simulación del sistema capturado le ofrece al usuario la posibilidad de observar cómo es que el sistema se comporta bajo ciertas entradas (marcado). El sistema SPADES ofrece dos formas de simulación, la manual y la automática .

La simulación manual espera la interacción del usuario, ya que es éste el que debe de disparar las transiciones que se encuentren habilitadas y de esta manera poder apreciar la evolución del marcado. Las transiciones que se encuentran habilitadas se distinguen de las que no lo están, por que se presentan en un color verde. Por su parte la simulación automática ejecuta una a una las transiciones que estén habilitadas, seleccionando cual dispara de manera aleatoria y así sucesivamente hasta que el propio usuario decida detener la simulación o no existan más transiciones que disparar.

La evolución del marcado en ambos casos se puede apreciar debido a la coloración en verde de las transiciones habilitadas, y a que se puede observar como las marcas pasan de un lugar a otro. Figura 4.18.

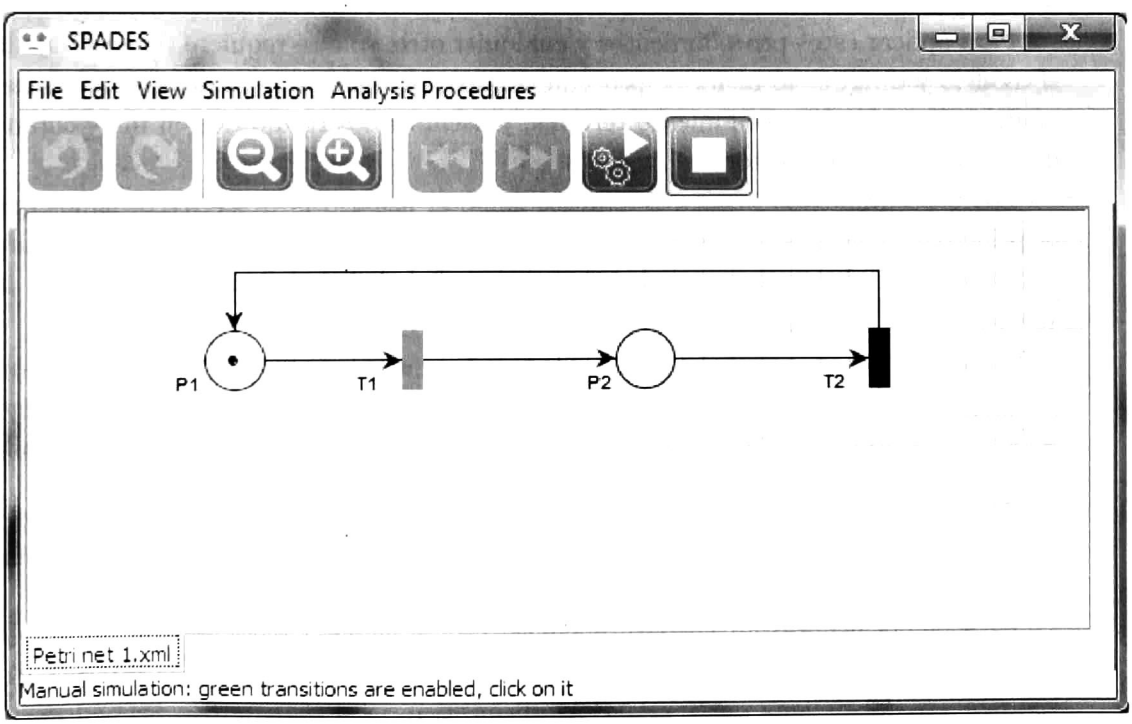


Figura 4.18: Simulación de un modelo dado.

La simulación puede ser iniciada de dos formas, mediante el botón de simulación en la barra de herramientas o a través del menú de simulación. A continuación la barra de dibujo se desaparece del editor para evitar que el usuario modifique la red y evitar inconsistencias. Al iniciar la simulación esta es manual, para dar inicio a la simulación automática puede hacerse a través del botón en la barra de simulación o mediante el menú de simulación.

Para detener cualesquiera de las dos formas de simulación, puede hacerse mediante el botón de detener  dispuesto en la barra de simulación o a través del menú de simulación.

4.5. Análisis de modelo

El análisis del modelo se refiere a aplicar un procedimiento de análisis. De manera inicial, el sistema solo posee la obtención de los invariantes y la obtención el marcado de la red de Petri.

Para aplicar estos procedimientos y cualquier otro, solo se requiere tener dibujado el modelo y dirigirse al menú de procedimientos y seleccionar cual de ellos aplicar al modelo. Cuando el procedimiento termine, presentará los resultados en una ventana independiente al del editor.

Los procedimientos pueden ser tantos como el usuario lo desee, recordemos que la principal característica de SPADES es que brinda la plataforma necesaria para implementar nuevos procedimientos de análisis. Esta funcionalidad se describe en la siguiente sección.

4.6. Adición de nuevos procedimientos de análisis

Así como se describió en los requerimientos y se especificó en la arquitectura del sistema SPADES-V2, la herramienta permite al usuario agregar nuevos procedimientos de análisis de una manera sencilla y segura para la herramienta.

Es decir, el usuario no necesita conocer todo el sistema para adicionar los procedimientos, sólo requiere conocer los métodos que le proporcionaran la información que requiere del sistema que modela. Al mismo tiempo el sistema fue diseñado para evitar el efecto domino que pudieran ocasionar la adición de dichos procedimientos, asegurando no afectar las otras funcionalidades ofrecidas por la herramienta.

Para lograr dicha funcionalidad, además de diseñar la arquitectura en base a las propiedades de modificabilidad y usabilidad, se hace uso del *Java Reflection API.*

4.6.1. Java Reflection API

Reflexión (Reflection) es la capacidad de un programa en ejecución para examinarse así mismo y su entorno de software, y para cambiar lo que hace en función de lo que encuentra. Para realizar esta autoexaminación, un programa debe contar con una representación de sí mismo. Esta información la llamaremos *metadatos.* En un mundo orientado a objetos, los metadatos se organizan en *objetos*, llamados *metaobjetos.* [8]

La autoexaminación en tiempo de ejecución de los metaobjetos se le llama ***Introspección***. Esta instrospección es seguida por un cambio de comportamiento. En general, hay tres técnicas que la reflexión puede utilizar para facilitar el cambio de comportamiento:

a La modificación directa del metaobjeto.

b Operaciones para el uso de metadatos (como la invocación dinámica de métodos).

c La intercesión, en la que se le permite al código interceder en las diversas fases de ejecución del programa.

El aporte del uso de la técnica de la reflexión al sistema SPADES es que ayuda a generar software Flexible, permite al sistema adaptarse fácilmente a las necesidades emergentes, además que apoya la *usabilidad* del sistema y la *reusabilidad* propiedades de calidad deseables en el sistema SPADES.

En sí, la reflexión realiza lo siguiente:

- Examina la estructura o los datos del programa.
- Tomar decisiones con los resultados de los pasos previos.

  Cambiar el comportamiento, la estructura, o los datos del programa basado en las decisiones que se hayan tomado.

4.6.2. Módulos y clases involucrados

Para que la adición de los nuevos procedimientos de análisis generados por el usuario sea de una manera sencilla y transparente, se generaron un conjunto de clases que se encargan de manipular el *reflection API* para acceder a los datos proporcionados en la clase, así como adicionar el procedimiento al menú correspondiente.

Este conjunto de clases son:

`ModuleRefelectManager.java`

Esta clase se encarga de gestionar todo lo referente al *Java Reflection API.*

`ModuleRemove.java`

ModuleRemove es la clase que se encarga de generar el menú *procedimientos de análisis*, así como adicionar los procedimientos a dicho menú, que el usuario genere De igual forma actualiza el estado del menú cada vez que el sistema se vuelve a ejecutar.

`ModuleLoad.java`

Esta clase obtiene el archivo fuente del procedimiento.

`ModuleResul.java`

Esta clase permite que los procedimientos puedan generar ventanas mediante las cuales muestren sus resultados.

Estas clases están contenidas dentro del subsistema *administrador de procedimiento* Vea Figura 4.19. Cada procedimiento que el usuario genere debe importar ciertas clases obligatorias definidas por el propio sistema SPADES las cuales le permitirán acceder a la estructura de la red de Petri y manipularla.

Así mismo cada procedimiento contiene un método que le permite a cada procedimiento ejecutarse cuando éste es invocado mediante el menú del sistema SPADES.

Lo anterior se observará de manera más clara en la siguiente sección.

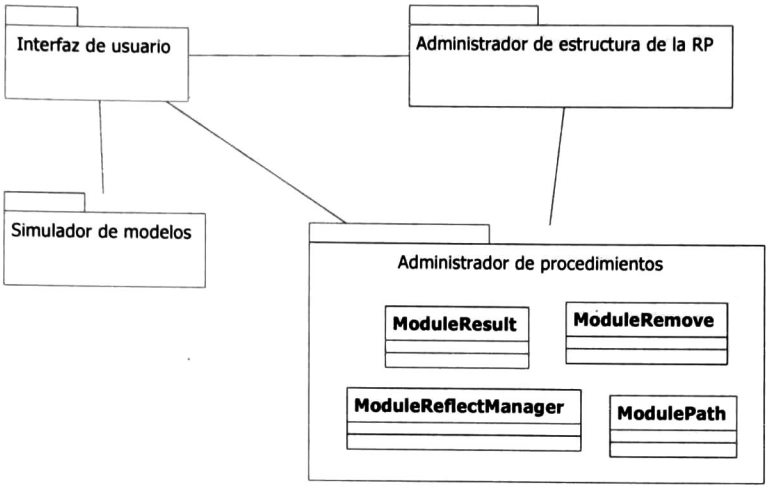


Figura 4.19: El subsistema *Administrador de procedimientos* y sus clases.

Figura 4.20: Ejemplo: carpeta para un procedimiento proc1.

4.6.3. Cómo adicionar un nuevo procedimiento de análisis

En esta sección se presenta los pasos que debe seguir el usuario para generar una nueva clase que contenga el procedimiento de análisis que quiera generar. Suponga que el procedimiento se llama ***proc1***.

- El primer paso es generar una carpeta que se llame de la misma forma que dese nombrar el procedimiento como en la Figura 4.20.
- Esta carpeta debe ser colocada en el siguiente directorio:

  `SPADES\structure\modules\`

  Independientemente de donde usted haya almacenado el sistema SPADES.
- Ahora debe generar una clase en el intérprete de Java con las siguientes características:

a Una linea que indique el nombre del paquete

```
package structure.modules.proc1;
```

b Importe el conjunto de paquetes o clases de Java que sean necesarias para su procedimiento. Para este ejemplo usaremos estas dos:

```
import java.awt.Container;
import javax.swing.BoxLayout;
```

c Ahora debe agregar el conjunto de clases que son obligatorias para el desarrollo del procedimiento *proc1*.

```
//estas tres primeras permiten obtener la estructura de la red de Petri.

import structure.dataLayer.DataLayer;
import structure.dataLayer.Place;
import structure.dataLayer.Transition;

//estas otras hacen posible que el usuario visualice los resultados que arrojará su procedimiento.

import structure.gui.CreateGui;
import structure.gui.widgets.ResultsHTMLPane;
import structure.gui.widgets.EscapableDialog;

//esta última es la más importante, debido a que permite que el procedimiento sea adicionado de manera automática al sistema SPADES.

import structure.modules.Module;
```

d Ahora debe comenzar la definición de la clase, propia del procedimiento *proc1*.

```
//la clase proc1 debe obligatoriamente implementar a la clase Module.

public class proc1 implements Module {

//una linea que indique el nombre del procedimiento.

      private static final String MODULE_NAME = "proc1";


//declaramos una variable de tipo ResultsHTMLPane que nos ayudará para visualizar los resultados.
```

```
private ResultsHTMLPane results;
```

//debemos generar siempre el método *run* ya que este nos permitirá que nuestro procedimiento *proc1* sea ejecutado.

```
public void run(DataLayer pnmlData){
```

//la siguiente linea permitirá que la venta de resultados indique el nombre del procedimiento.

```
EscapableDialog guiDialog =
        new EscapableDialog(CreateGui.getApp(),
                    MODULE_NAME, true);
```

//las siguientes lineas permiten relacionar la ventana de resultados al editor

```
Container contentPane = guiDialog.getContentPane();
contentPane.setLayout(new BoxLayout
            (contentPane, BoxLayout.PAGE_AXIS));
```

//creamos una variable *data* de tipo *DataLayer* la cual permitirá el acceso a la estructura de la red de Petri que el usuario haya capturado en el editor gráfico

```
DataLayer data = CreateGui.currentPNMLData();
```

//una variable de tipo *Sring* mediante la cual manejaremos el texto y los mensajes que se visualizaran en la ventana de resultados

```
String texto = "";
```

//verificamos si existe un tab abierto

```
if (data == null) {
   return;
}
```

//en la siguiente sentencia de condición *if else* el usuario agregara el código de su procedimiento, es decir las lineas que éste necesite para generar la funcionalidad de su procedimiento

//si no existe una red de Petri dibujada en el editor, a *texto* le adicionamos el letrero que queremos que aparezca en la ventana de resultados, en este caso *No existen datos en el editor Dibuje o abra una red existente*

```
if (!data.hasPlaceTransitionObjects()) {
   texto + = "<h2>No existen datos en el editor<br>" +
      "Dibuje o abra una red existente</h2>";
}
```

//en caso de que si exista una red, aquí es donde se deben de agregar las lineas que definen al procedimiento *proc1* para este ejemplo sólo imprimiremos el marcado de cada lugar

```
else{
   int[] arr = data.getMarking();
   for( int x=0; x<data.getPlacesCount();x++){
      texto + =  "<br> Marcado P" + x +
      + Integer.toString(arr[x]) + "<br>";
   }
}
```

//esta linea sirve para dar formato al texto de la ventana de resultados

```
results.setText(s);
}
}
```

- después de haber generado la clase, ésta debe almacenarla en la carpeta que se generó como primer paso. Figura 4.20
- Ahora sólo hace falta compilar *proc1* para que sea agregado al sistema SPADES. Para ello requiere inicializar una ventana de comandos y dirigirse al directorio hasta donde esta su procedimiento.

```
.../SAPDES/structure/modules/proc1
```

- Ahora debe teclear el comando *javac* el cual le permitirá compilar su clase, para nuestro ejemplo *proc1.java*

  ```
  ./proc1 javac proc1.java
  ```

  Si su código tiene errores estos aparecerán en la ventana de comandos, debe corregirlos y volver a realizar este paso

- En caso contrario, debe hacer lo siguiente:

  ```
  ./SPADES java BuildGui
  ```

  Esto ejecutará al sistema SPADES y *proc1* aparecerá en el menú de *Procedimientos de Analsisis*, y el usuario podrá comenzar a utilizarlo.

Para agregar otro nuevo procedimiento sólo se necesita volver a realizar los pasos anteriormente mencionados.

4.6.4. Eliminar un procedimiento de análisis

Puede darse el caso que el usuario ya no desee mas un procedimiento de análisis, por lo que el usuario tiene la opción para eliminarlo del sistema. Para ello requiere de ejecutar unos simples pasos.

Supongamos que el procedimiento a eliminar tiene por nombre *proc1*.

- Diríjase al siguiente directorio:

  ```
  ./SPADES/structure/modules/
  ```

- Ahí busque la carpeta que contiene a dicho procedimiento y que lleva el mismo nombre. Vea Figura 4.20

- Selecciónela y presione suprimir, o elimínela de la forma que usted acostumbra o gusta de borrar documentos.

- Ejecute SPADES y diríjase al menú de *procedimientos de análisis.*

  ```
  .../SPADES java BuildGui
  ```

  Podrá observar que ya no existe el procedimiento en el menú. También puede quitar la carpeta del directorio y pasarla a otro lugar, esto tendrá el mismo efecto,

4.7 Presentación de algunas de las clases que forman el sistema SPADES

4.7.1. Clases que definen y permiten manipular la red de Petri

En esta sección se describen las clases que forman la definición de la estructura de la red de Petri, así como algunos de sus métodos para manipular los datos del elemento que define la clase.

Arc

Esta clase contiene la definición del arco (Identificador,Posición inicial en el eje X, Posición inicial en el eje Y, Posición final en el eje X, Posición final en ele eje Y, Nodo fuente, nodo destino, Peso).

La clase contiene métodos que permiten obtener el nombre del arco *getName()*, el nodo fuente al que esta conectado *getSource()*, el nodo destino al que esta conectado *getTarget()*, el peso asignado al arco *getWeight()*, obtener los puntos que forman el camino del arco *getArcPath()*, entre muchos métodos más.

Transition

Una transición se define mediante su Identificador, la posición en el eje X, la posición en el eje Y, asi como su angulo de inclinación.

Esta clase contiene métodos que permiten conocer si la transición se encuentra habilitada *isEnabled()*. Métodos que permiten hacer una copia *copy()*, pegarla *paste()* o eliminarla del editor *delete()*. *getAngle()* permite obtener el ángulo de inclinación de la transición, mientras que *getName()* proporciona el nombre de la transición.

Place

Al igual que las clases anteriores, esta clase contiene la definición de un lugar, el cual es definido mediante el Identificador, posición en el eje X, posición en el eje Y, y marcado inicial.

Los métodos que permiten manipular estos datos son *getName()* el cual proporciona el nombre del lugar, *getInicialMarking()* proporciona el marcado inicial mientras que *getCurrentMarking()* proporciona el marcado actual, el cual es utilizado durante

la simulación. También esta clase define métodos para copiar *Copy()*, pegar *Paste()* y eliminar *Delete()*, entre otros.

DataLayer

Esta clase encapsula a la red de Petri en su totalidad. Contiene métodos que permiten crear cualquiera de los elementos de la red (*createArc*,*createTransition*, *createPlace*). También permite obtener un arreglo con todos los lugares *getPlaces*, o un arreglo con todos los arcos *getArcs*, o un arreglo con todas las transiciones *getTransitions* que se hayan capturado.

Mediante los métodos *addArc*,*addTransition* y *addPlace*, esta clase mantiene la relación de lo elementos capturados por el usuario en el editor.

Esta clase también contiene métodos que permiten obtener el marcado actual de la red (*getMarking()*), asi como la matriz de incidencia (textitgetIncidenceMatrix()), pre (*getBackwardsIncidenceMatrix*) y post (*getForwardsIncidenceMatrix*).

4.7.2. Clases que permiten editar los atributos de la red de Petri

En esta sección se describen las tres clases que permiten modificar los atributos de una red de Petri.

EditWeightAction

Esta clase permite al usuario cambiar el peso del arco seleccionado. EditWeightAction es una clase muy pequeña que solo contiene el constructor correspondiente para realizar dicha modificación. Por default, todo arco tiene como peso 1. Si se desea agregar mas atributos, en esta clase deben de ser agregados los métodos correspondientes que describan dichos atributos.

PlaceEditor

Esta clase permite crear la ventana que permite modificar los atributos del lugar seleccionado. En esta clase se definen las etiquetas y cajas de texto, que permitirán al usuario introducir los datos que desee modificar. Actualmente los atributos que pueden ser modificados por el usuario son el nombre del lugar, así como su marcado.

Esta clase hace uso del archivo *PlaceEditorForm.form*, que no es más que el archivo de la forma utilizada para implementar la ventana.

Los nuevos atributos que deseen adicionarse, deben hacerse en el archivo *PlaceEdito.java* (los métodos y generación de las cajas de texto o etiquetas necesarias para solicitar el datos al usuario), y en el archivo *PlaceEditorForm.form*.

TransitionEditor

Esta clase funciona de la misma manera que la clase *PlaceEditor*, solo que ésta permite crear la ventana para modificar los atributos de la transición seleccionada.

Esta clase también hace uso de un archivo .form el *TransitionEditorForm.form*, que es el archivo que describe la forma utilizada para la implementación de la ventada de edición de los atributos de la transición.

Los atributos nuevos que se deseen agregar se tienen que hacer en ambos archivos *PlaceEditor* y TransitionEditorForm.form

Capítulo 5

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones del desarrollo de esta tesis, así como el trabajo futuro para el sistema SPADES.

5.1. Conclusiones

En esta tesis se desarrolló una nueva versión de la herramienta SPADES. Se utilizó el lenguaje Java para permitir que el sistema pudiera ser ejecutado en diferentes plataformas, evitando así, limitar al usuario al uso de un sistema operativo o plataforma en específico. Otra razón por la que se eligió el lenguaje Java es por la funcionalidad del *Java Reflection API*, lo cual ayudó a desarrollar la parte más importante del sistema: permitir al usuario agregar procedimientos de análisis.

Durante el desarrollo del sistema SPADES, se hizo uso de la *Ingeniería de Software* para la *Especificación de los requerimientos* apoyándose en el Lenguaje de Modelado Unificado (UML) para complementar la especificación mediante los *diagramas de caso de uso* y de los *diagramas de secuencia* que ayudaron a definir en una etapa temprana la interacción del usuario con el sistema, así como los elementos que conforman el sistema y la comunicación entre ellos para ofrecer las distintas funcionalidades al usuario.

Para el diseño de la arquitectura se hizo uso del patrón arquitectónico Modelo-Vista-Controlador (MVC), el cual permitió adaptar los objetivos de calidad requeridos por el sistema SPADES. Como MVC divide el diseño de la arquitectura en tres partes, esto ayudó a modularizar el sistema, tratando de generar clases que definieron una funcionalidad en específico para evitar el traslape de funcionalidades y reducir efectos secundarios cuando se realicen modificaciones.

Así mismo, durante el diseño de la arquitectura se hizo uso de la metodología ADD (Attribute-Driven Desing), la cual mediante la descomposición recursiva de los *Architectural Drivers* ayudó a especificar una única funcionalidad por subsistema.

En esta tesis se logró desarrollar una herramienta computacional que permite la captura de modelos que describen Sistemas de Eventos Discretos (SED) mediante el uso de redes de Petri como formalismo de representación y análisis. SPADES cuenta con un *Editor Gráfico*, *Simulación* automática y manual, y las facilidades para que el usuario agregue nuevos procedimientos de análisis así como nuevas extensiones de redes de Petri.

5.2. Trabajo Futuro

Actualmente la herramienta SPADES permite la obtención de los *Invariantes* y el cálculo de la Matriz de Incidencia como procedimientos de análisis. Por lo que generar una cantidad mayor de procedimientos y adicionarlos a la herramienta sería una tarea importante.

La herramienta solo permite representar redes de Petri ordinarias, por lo que generar nuevas extensiones de redes de Petri permitiría al usuario capturar una clase más amplia de sistemas. Si esto llegará a desarrollarse, se podría presentar un menú de opciones en el que el usuario pueda elegir cuál tipo de red de Petri desea capturar.

Actualmente no se cuenta con algún elemento que permita clasificar los procedimientos de análisis, por lo que adicionarlo, aumentaría la usabilidad de la herramienta, en el aspecto a la aplicación de un procedimiento al sistema que haya sido capturado por el usuario.

Al momento que el usuario adicione procedimientos de análisis o extensiones de redes de Petri, debería de generar automáticamente la documentación de las clases y métodos que modifique o agregue a la herramienta.

Bibliografía

[1] Avgeriou, P. and U. Zdun (2005). Architectural patterns revisited a pattern language. In *Proceedings of 10th European Conference on Pattern Languages of Programs*, pp. 1–39.

[2] Bass, L., P. Clements, and R. Kazman (2005). *Software Architecture in Practice: Second Edition.* Addison-Wesley.

[3] Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal (1996). *Pattern-Oriented Software Architecture: A System of Patterns.* Wiley & Sons.

[4] Cassandras, C. G. and S. Lafortune (1999). *Introduction to Discrete Event Systems.* Kluwer Academic Publishers.

[5] Chaouiya, C. (2007). Petri net modelling of biological networks. *Briefings in Bioinformatics*, 210–219.

[6] Coutaz, J. (1987). Pac: an implementation model for dialog design. In *Proceedings of the Interact'87 conference*, pp. 1–4.

[7] Deitel, H. M. and P. J. Deitel (2008). *Java How to Program.* Prentice Hall.

[8] Forman, I. R. and N. Forman (2005). *Java Reflection in Action.* Manning.

[9] Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1997). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

[10] Group-TGI (2008). Petri nets world.

[11] Ho, Y. C. (1989). Special issue on dynamics of discrete event systems. In *Proceedings of the IEEE*, pp. 1.

[12] Jürgen, W. and M. Heiner (2002). Petri nets in biology, chemistry, and medicine. Technical report, Brandenburg University of Technology at Cottbus.

[13] Liang, Y. D. (2005). *Introduction to Java Programming.* Addison-Wesley.

[14] Murata, T. (1989). Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, Volume 77, pp. 541–580.

[15] Niemeyer, P. and J. Knudsen (2005). *Learning Java.* O'Reilly.

[16] Reenskaug, T. (1979a). Models views controllers. Technical report, Xerox PARC.

[17] Reenskaug, T. (1979b). Thing-model-view-editor, an example from a planning-system. Technical report, Xerox PARC.

[18] Reenskaug, T. (2003). The model-view-controller (mvc), its past and present. Technical report, Xerox PARC.

[19] Rozanski, N. and E. Woods (2005). *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives.* Addison-Wesley Professional.

[20] Schildt, H. (2007). *Swing: A Beginner's Guide.* Osborne Mcgraw Hill.

[21] Shaw, M. and D. Garlan (1996). *Software Architecture: Perspectives on an Emerging Discipline.* Prentice-Hall.

[22] Sommerville, I. (2005). *Ingeniería del software* (7a ed.). Pearson.

[23] Störrle, H. (1998). An evaluation of high-end tools for petri nets. Technical report, Ludwing-Maximilians Universität München Institut für Informatik.

[24] Taylor, R. N., N. Medvidovic, E. M. Dashofy, and E. M. Dashofy (2010). *Software Architecture: Foundations, Theory, and Practice.* John Wiley & Sons, Inc.

[25] Wiegers, K. E. (2003). *Software Requeriments:Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle.* Microsoft Press.

[26] Wiegers, K. E. (2006). *More About Software Requirements: Thorny Issues and Practical Advice.* Microsoft Press.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.

UNIDAD GUADALAJARA

"2010, Año de la Patria, Bicentenario del Inicio de la Independencia y Centenario del Inicio de la Revolución"

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Una herramienta para el análisis de modelos en redes de Petri

del (la) C.

Ana Karina RÍOS ARAUJO

el día 20 de Septiembre de 2010.

Dra. Ofelia Begóvich Mendoza
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Antonio Ramírez Treviño
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González Pico
Investigador CINVESTAV 2A
CINVESTAV Unidad Guadalajara