Universidad de las Ciencias Informáticas

Facultad 3

Complemento para transformar un Árbol de Variantes en una red de Petri

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Yoelvis Ortiz Chaviano

Tutora: Ing. Dina Yaksilik Torres Sakipova

Co-tutor: Ing. Damián Pérez Alfonso

La Habana, Julio de 2014

"Año 56 de la Revolución"

PENSAMIENTO

"No es tarea de la Universidad ofrecer lo que la sociedad le pide, sino lo que la sociedad necesita"

Edsger Wybe Dijkstra

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste se firma la presente a los _____ días del mes de _____ del año _____.

_____

Yoelvis Ortiz Chaviano

_____ _____

Ing. Dina Yaksilik Torres Sakipova Ing. Damián Pérez Alfonso

AGRADECIMIENTOS

A mis padres por su ayuda, apoyo incondicional y por siempre tener fe en mí.

A mi hermano Yidelvis por darme fuerzas y esperar de mí lo mejor.

A mis tutores Dina y Damián por su tiempo, guía y sobre todo por exigirme tanto para que este momento se hiciese realidad.

Al tribunal por sus críticas constructivas, por sus sugerencias y por su ayuda.

A todos mis compañeros del aula, en especial a Yasel, Pimentel, Javier, Alejandro, el cuadro, Cervetto, Yoalvy, Rainer, Osmail, Sanamé, Ledián, Felix y René.

A Sary, Wanda, Jessica, Isamira y Lili por su amistad y apoyo incondicional.

A mis amigos Gabriel y Eduardo por incitarme a coger esta carrera y por tratarme como parte de su familia.

A Leandro por su apoyo incondicional y por ser como un hermano para mí.

A Galafet por aconsejarme siempre, por ser la mejor profesora que he tenido, por ser esa persona maravillosa que la caracteriza, por sacarme de buenos aprietos y por hacer la mejor sopa del mundo.

A todos mis amigos de la UCI y del IPVCE.

A todos que de una forma u otra hicieron de mi paso por la universidad una experiencia inolvidable.

Yoelvis…

DEDICATORIA

A mis padres, por la confianza que tienen en mí, por esa dedicación y amor que me profesan cada día.

A mi sobrinita, por ser tan linda y por darme tantas fuerzas cada vez que dice.....te quiero tío!!!

A mi familia en general, en especial a mi hermano Yidelvis por siempre tenerme en cuenta y por guiarme en la vida.

A todos mis amigos por darme su apoyo y por comportarse como verdaderos hermanos. Yoelvis…

RESUMEN

La mayoría de las empresas analizan, gestionan y mejoran sus procesos, en su gran mayoría apoyadas por sistemas informáticos que generan registros de eventos. Las técnicas y herramientas desarrolladas en la Minería de Procesos permiten extraer de los registros de eventos de procesos poco estructurados información valiosa, pero presenta numerosos problemas con el tratamiento del ruido y la completitud. La técnica Minería de Variantes corrige estas dificultades, pero presenta la limitante que devuelve un modelo jerárquico en forma de árbol, que no permite aplicar métricas para medir la calidad de estos modelos, que en su mayoría son aplicables a redes de Petri. Por ello se desarrolló un complemento que transforma un Árbol de Variantes en una red de Petri. De esta manera se tiene un modelo equivalente en notación red de Petri, al cual es posible aplicar las métricas de calidad de los modelos en la Minería de Procesos. Para su desarrollo se utilizó el Entorno de Desarrollo Integrado Eclipse, con el que se creó el complemento para ProM. Para validar la propuesta de solución se consultaron especialistas que valoraron el funcionamiento del complemento, es decir, compararon la equivalencia entre un Árbol de Variantes y una red de Petri.

Palabras claves: Minería de Proceso, procesos poco estructurados, Minería de Variantes, Árbol de Variantes, red de Petri.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El acelerado desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC), ha propiciado el perfeccionamiento tecnológico de empresas, organismos gubernamentales y organizaciones. Actualmente los sistemas informáticos son una pieza importante en el funcionamiento de fábricas, centros de investigación, agencias de viajes y transporte de cargas, bancos, hospitales, estaciones de policías, oficinas y entidades en general. La mayor parte de las tareas realizadas en estas instituciones se ejecutan y controlan por sistemas de información con el fin, en algunos casos, de almacenar datos en forma de trazas y utilizarlos como apoyo al control y la toma de decisiones.

Las empresas actuales dan crédito a la ventaja que supone agregar nuevas tecnologías en sus procesos de negocio. Esta idea promueve el uso de diferentes herramientas diseñadas para la gestión de dichos procesos, que en muchos casos, registran trazas del funcionamiento y la información generada durante determinadas actividades empresariales. Ejemplo de estas herramientas son las usadas en la Gestión de Procesos de Negocio (Magliano, y otros, 2013). El aprovechamiento de las trazas que contienen información de los registros de eventos para analizar los procesos de negocio, es el objeto de una nueva área de investigación: la Minería de Procesos.

La Minería de Procesos se encarga de descubrir, monitorear y mejorar los procesos reales que se llevan a cabo en una empresa, extrayendo conocimiento de los registros de eventos comúnmente disponibles en los sistemas informáticos actuales (van der Aalst, y otros, 2011).

Cuba está incursionando desde hace varios años en la informatización de sus organismos y empresas, aportando nuevas ideas y conocimientos en el intento de encontrar una solución a los problemas que surgen durante el constante y necesario perfeccionamiento de estas instituciones. En septiembre del 2002 se fundó la Universidad de las Ciencias Informáticas (UCI) con la misión de ayudar a informatizar la sociedad cubana. Precisamente en la UCI, se ha creado un grupo de investigación en la Facultad 3, que se encarga de estudiar la Minería de Procesos.

Esta rama parte de las trazas de los registros de eventos y en muchas ocasiones contienen actividades ejecutadas en diversos órdenes, donde varía mucho el momento en el que se ejecutaron, las actividades precedentes y posteriores cambian, cambian los autores, o se evidencian muchas relaciones o variaciones entre las actividades (van der Aalst, 2011).

Estos procesos conocidos como procesos poco estructurados generan modelos poco comprensibles, que no aportan información debido al alto grado de dificultad que asume comprenderlos (Günther, 2009) (Chandra Bose, y otros, 2010).

Existen muchas técnicas desarrolladas hasta el momento para obtener modelos de este tipo de procesos, tales como Minería Difusa y Alineación de Trazas; pero presentan problemas con el tratamiento del ruido (ausencia de información), el número de tareas involucradas, las dimensiones y/o la complejidad del registro de eventos. La técnica Minería de Variantes, recientemente desarrollada, trata de disminuir estas dificultades. Para ello genera la variante más óptima del proceso, donde una variante es el camino que va desde el inicio hasta el final del proceso. Inconvenientemente, esta técnica tiene como salida un modelo jerárquico, es decir, en forma de árbol (Alfonso, y otros, 2014) lo que no permite que se le aplique métricas existentes para medir la calidad de los modelos.

Sin embargo, existe una notación nombrada Redes de Petri muy utilizadas en técnicas de descubrimientos de procesos como Minería Alfa y Minería ILP (Programación Lineal Entera). A esta notación le es aplicable una gran cantidad de métricas de calidad tales como Aptitud, Precisión ETC (Técnica de Evaluación de Conformidad, del inglés Evaluation Tecnique Conformance), las cuales comprueban qué proporción del comportamiento del registro de eventos es capturado de manera correcta y qué tan exacta es esa captura del comportamiento con respecto al registro de eventos (Broucke, y otros, 2013).

El hecho de que se puedan aplicar estas métricas, incide en que si el modelo muestra correctamente los procesos contenidos en un registro de eventos, la información extraída es fidedigna, por lo tanto, ayuda a la detección de fraude, anomalías, procesos incompletos, y a comprender el verdadero funcionamiento de los procesos en empresas y organizaciones. De ahí la importancia de la aplicación de las métricas de calidad a un árbol de variantes.

A partir de la problemática planteada se define como **problema a resolver:** ¿Cómo transformar un árbol de variantes a una red de Petri de manera que facilite aplicar métricas para medir su calidad?

Constituye el **objeto de estudio:** Técnicas de descubrimiento de procesos.

Se define como **objetivo de la investigación:** Desarrollar un algoritmo que transforme un árbol de variantes en una red de Petri para poder aplicarle las métricas existentes para medir la calidad del modelo.

Se identifica como **campo de acción:** Modelado de procesos con redes de Petri en el descubrimiento de procesos.

Se plantea como **idea a defender:** Si se desarrolla un algoritmo que transforme un árbol de variantes en una red de Petri, se le podrá aplicar las métricas que existen en la actualidad para medir la calidad del modelo.

Se desglosan del objetivo general los siguientes **objetivos específicos:**

- Realizar la fundamentación teórica de la investigación.
- Desarrollar un algoritmo que transforme un árbol de variantes en una red de Petri.
- Validar la solución propuesta para comprobar el cumplimiento del objetivo.

Para dar cumplimiento a los objetivos propuestos se definen las siguientes **tareas de investigación:**

- Estudio de los principales conceptos y temas relacionados en la Minería de Procesos.
- Estudio de los tipos de minería, así como de los patrones de control de flujo de los procesos.
- Análisis de las principales dificultades de los modelos de procesos poco estructurados.
- Análisis de los algoritmos y técnicas existentes para la visualización de los procesos poco estructurados.

- Estudio de las métricas existentes para medir calidad y rendimiento de las técnicas y modelos descubiertos en la Minería de Procesos.
- Estudio de algunas notaciones para modelar procesos.
- Estudio de la herramientas que se utilizan en la Minería de Procesos para generar los modelos de procesos.
- Diseño de las reglas de transformación para generar el modelo seleccionado a partir de un árbol de variantes.
- Implementación del algoritmo diseñado (como una forma de validación del mismo).
- Evaluación de la herramienta desarrollada a través de pruebas.

La realización de dichas tareas se sustenta en los siguientes métodos de investigación:

Métodos Teóricos:

- **Analítico – sintético:** Permite realizar el estudio de las necesidades y los requisitos identificados, sintetizar las ideas para lograr el desarrollo de un complemento que brinde la solución al problema planteado.
- **Histórico-lógico:** Permite conocer la evolución de los diferentes conceptos y tendencias referentes al objeto de estudio de la investigación, y de esta forma obtener un mayor conocimiento y saber aplicarlo en la solución del problema presentado.
- **Modelación:** Permite trabajar con los modelos generados a partir de los registros de eventos y de la propia solución, además de los diagramas y artefactos que genera la metodología utilizada.

Métodos Empíricos:

- **Medición:** Permite aplicar métricas para medir la calidad del diseño de clases propuesto, y de los modelos de procesos.

La investigación está estructurada en tres capítulos:

Capítulo 1: Fundamentación teórica.

En este capítulo se recogen los principales aspectos teóricos de la investigación. Se describen los principales conceptos de la Minería de Procesos. Se realiza un análisis de

los procesos estructurados y poco estructurados, así como de las principales técnicas de diagnósticos de procesos. Se describen la metodología y las herramientas que se utilizaron para darle solución al problema.

Capítulo 2: Propuesta de solución.

En este capítulo se muestra una técnica que convierte un árbol de variantes a su equivalente en una red de Petri. Se presentan los artefactos generados por la metodología utilizada para darle solución al problema. Se valida el diseño empleado con las métricas Tamaño Operacional de Clase y Relaciones entre Clases.

Capítulo 3: Validación de la solución.

Se realiza la descripción de las pruebas utilizadas para validar la solución. Se aplican pruebas de caja blanca y casos de pruebas funcionales. Se aplican métricas de calidad a los modelos obtenidos con la solución. Se hace una consulta a especialistas para comprobar el cumplimiento de los objetivos propuestos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se explorarán los principales conceptos de Minería de Procesos. Se describen las principales técnicas que generan modelos de procesos, entre ellas la técnica Minería de Variantes, y sus principales deficiencias. Se propone el uso de un algoritmo que convierta un Árbol de Variantes de la técnica Minería de Variantes en una red de Petri.

1.2. Minería de Procesos.

Un proceso de negocio consiste en un conjunto de actividades que se realizan en coordinación en un entorno organizativo y técnico. Cada proceso de negocio es ejecutado por una sola organización, pero puede interactuar con los procesos de negocio llevados a cabo por otras organizaciones (Weske, 2007).

Las empresas y organizaciones al realizar actividades en sus procesos de negocio cada vez más complejas e interrelacionadas hacen que los procesos sean confusos y generen mayor volúmenes de datos e información. De ahí la importancia de asistir la gestión de sus procesos de negocios utilizando sistemas de información como: Planificación de Recursos Empresariales (ERP, del inglés Enterprise Resource Planning), los de Gestión de Procesos de Negocio (BPM, del inglés Business Process Management). Estos sistemas proveen herramientas tales como Openbravo ERP (Openbravo, 2007) y Ultimus BPM suite (Ultimus, 2012), que guardan en trazas la ejecución de las actividades de las empresas y organizaciones. Para poder extraer información de estas trazas es que surge la Minería de Procesos.

En su libro (van der Aalst, 2011) van der Aalst plantea que "*la idea de la Minería de Procesos es descubrir, monitorear y mejorar los procesos reales mediante la extracción de conocimiento a partir de los registros de eventos fácilmente disponibles en los sistemas de hoy*".

La Minería de Procesos consiste en general, en obtener modelos de procesos a partir de registros de eventos. Un registro de eventos es un conjunto de instancias de procesos de

un proceso de negocio. Una instancia de proceso se manifiesta como una traza. Una traza se define como una lista ordenada de actividades que invoca una instancia de proceso desde el inicio de su ejecución hasta el final (van der Aalst, y otros, 2009).

1.2.1. Procesos estructurados.

Los procesos, en el área de la Minería de Procesos, se dividen en dos grandes grupos: procesos estructurados o lasaña y procesos poco estructurados o espaguetis.

Cuando los registros de eventos muestran que los procesos se caracterizan por una manera de ejecutarse bastante controlada y que poseen una clara estructura, dichos procesos se denominan estructurados o lasaña. En este tipo de procesos todas las actividades se identifican con facilidad, además de tener las entradas y las salidas bien definidas. Los modelos generados a partir de los procesos tipos lasaña son sumamente comprensibles (van der Aalst, 2011).

Es importante añadir que los procesos estructurados, en general, se les puede aplicar la gran mayoría de las técnicas de Minería de Procesos, por lo que las mejoras que se pueden añadir son pequeñas debido a que están organizados y que los modelos que los representan son comprensibles (van der Aalst, y otros, 2011).

Esto significa que los procesos estructurados son frecuentemente menos interesantes, por lo que no es de interés abordar en la investigación. Sin embargo muchos procesos son menos estructurados de lo que se piensa, de forma que se dificulta modelarlos y comprenderlos.

1.2.2. Procesos poco estructurados.

Cuando los procesos tienen un alto grado de relación y se ejecutan desorganizadamente, mostrando una estructura confusa, son llamados procesos poco estructurados o procesos espaguetis.

Las técnicas actuales de Minería de Procesos que normalmente se esfuerzan por mostrar todo el comportamiento observado de los procesos, una vez que son aplicadas en estos procesos espaguetis, tienden a producir modelos complejos de poco entendimiento.

Estos procesos resultan más atractivos para las investigaciones actuales, pues presentan un amplio margen de mejora. Según (van der Aalst, y otros, 2009), en la actualidad no todos los negocios poseen sus procesos bien estructurados, por lo que los procesos tienden a ser menos estructurados de lo que se espera.

1.2.3. Tipos de Minerías de Procesos.

Los registros de eventos pueden ser utilizados para realizar tres tipos de Minería de Procesos (van der Aalst, y otros, 2011) (van der Aalst, 2011) (Magliano, y otros, 2013):

- **Descubrimientos de procesos:** Este tipo de minería es la más destacada y donde se ha centrado la mayoría de los esfuerzos. En el tipo descubrimiento se toma un registro de eventos y se produce un modelo sin usar ninguna información a-priori (van der Aalst, y otros, 2011).
- **Verificación o chequeo de conformidad:** En este tipo de minería se compara un modelo de proceso existente con un registro de eventos del mismo proceso. La verificación de conformidad puede ser usada para chequear si la realidad, tal como está almacenada en el registro de eventos, es equivalente al modelo y viceversa. La comprobación de la conformidad puede ser utilizada para detectar, localizar y explicar las desviaciones, así como medir su gravedad (van der Aalst, y otros, 2011) (van der Aalst, 2011).
- **Mejoramiento:** el propósito de este tipo de minería es extender o mejorar un modelo de proceso existente usando la información acerca del proceso real almacenado en un registro de eventos. Mientras la verificación de conformidad mide el alineamiento entre el modelo y la realidad, este tercer tipo de Minería de Procesos busca cambiar o extender el modelo a-priori. Un tipo de mejora es la reparación, es decir, modificar el modelo para reflejar mejor la realidad. Por ejemplo, si dos actividades se modelan de forma secuencial, pero en realidad pueden suceder en cualquier orden, entonces el modelo puede ser corregido para reflejar esto. Otro tipo de mejora es la extensión, es decir, la adición de un nuevo punto de vista para el modelo de proceso (van der Aalst, y otros, 2011) (van der Aalst, 2011).

En la Figura 1 se muestra cómo pueden ser utilizados los registros de eventos para realizar los tres tipos de minerías de procesos.

Figura 1: Representación de los tres tipos de minería de procesos (van der Aalst, 2011)

1.3. Técnicas de diagnóstico de procesos.

Las técnicas de Minería de Procesos permiten generar modelos que pueden ofrecer valiosas interpretaciones sobre cómo se ejecutan los procesos en la vida real (van der Aalst, y otros, 2009).

1.3.1. Diagrama de puntos (del inglés Dotted char).

El diagrama de puntos es una técnica de visualización de las instancias contenidas en el registro de eventos, que son descritas como puntos en un plano donde una dimensión hace referencia a los casos en el registro de eventos, y la otra dimensión se refiere al tiempo en que se ejecutaron estos. Se utiliza un color distinto para cada actividad que se encuentre contenida dentro del registro de eventos (Song, y otros, 2007) (van der Aalst, 2011). La Figura 2 es un ejemplo dela técnica diagrama de puntos en la cual se visualiza las instancias contenidas en un registro de eventos.

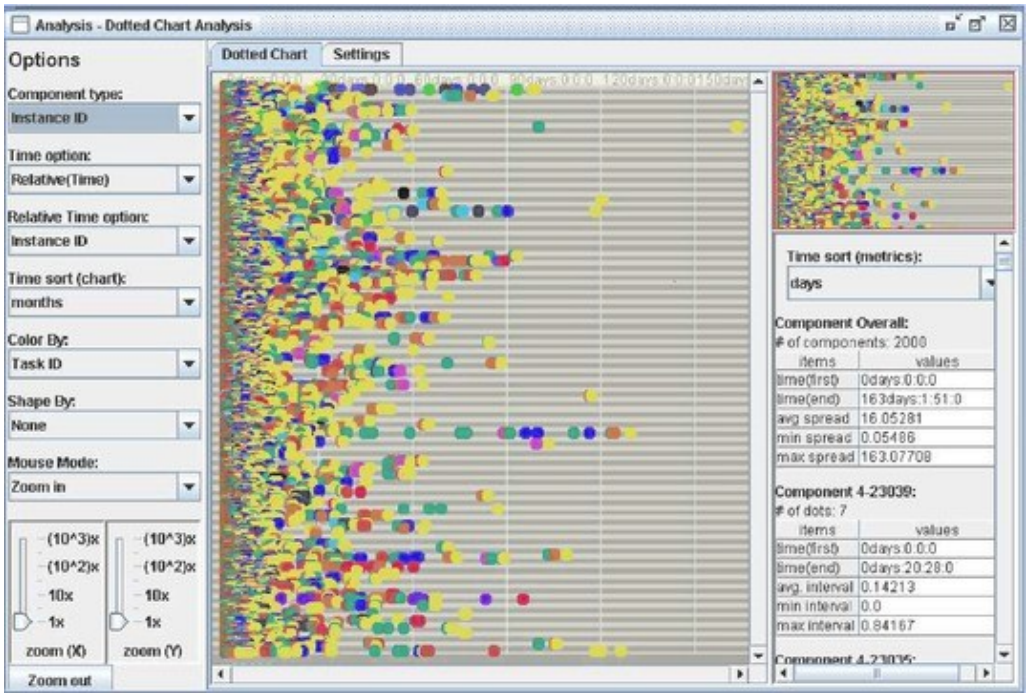


Figura 2: Diagrama de puntos de un registro de eventos con 3517 eventos (Song, y otros, 2007)

Se puede notar que en esta forma de graficar, los puntos o eventos no se encuentran alineados, por lo tanto es muy difícil determinar patrones comunes entre casos distintos.

1.3.2. Alineación de trazas (del inglés Trace Alignment).

La técnica alineación de trazas propuesta por Chandra Bose y van der Aalst facilita el problema de determinar patrones interesantes en medianos y grandes registros, que se determinan automáticamente y se muestran al usuario. El orden establecido entre las tareas permite identificar los patrones control de flujo que se manifiestan en el proceso. Esta técnica utiliza la programación dinámica para tabular las trazas de forma tal que se simplifiquen los problemas de entendimiento de las relaciones entre las actividades (Chandra Bose, y otros, 2010). En la Figura 3 se muestra un ejemplo del modelo que se obtiene al aplicar la técnica de alineación de trazas.

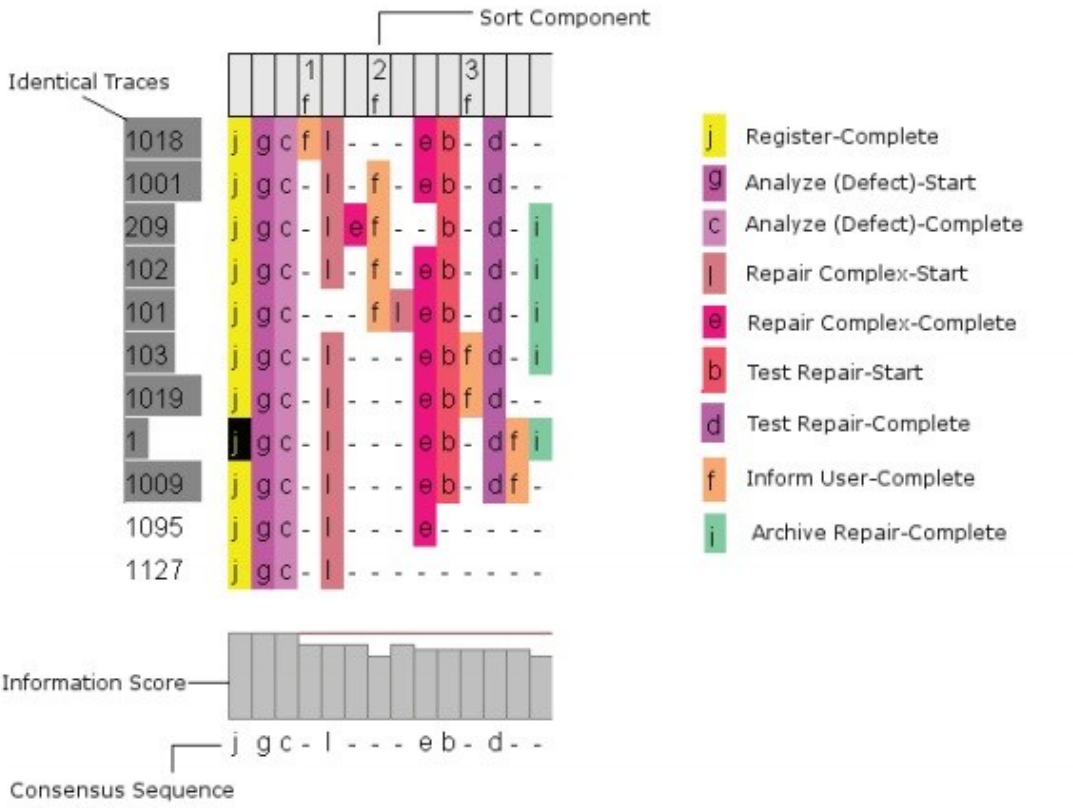


Figura 3: Técnica de Alineación de trazas (Chandra Bose, y otros, 2010)

No obstante, cuando hay ruido en las trazas el resultado puede ser de baja calidad. Por consiguiente, se necesitan técnicas que identifiquen y desechen el ruido durante la alineación.

1.3.3. Minería difusa (del inglés Fuzzy Mining).

Con la aplicación de esta técnica se obtiene un modelo basado en gráficos capaz de proporcionar una vista de alto nivel de un proceso, con la abstracción de los detalles no deseados. Está caracterizada por la presencia de dos tipos de nodos, que se refieren a una tarea y los que se refieren a un conjunto de tareas o clúster. No pueden ser usados para controlar la ejecución de los procesos debido a que no definen la lógica del mismo de una manera lo suficientemente precisa (Günther, 2009). Esto es intencional, ya que su objetivo es proveer una simple visualización del proceso, por tanto sacrifica precisión por comprensión y simplicidad. Además, esta técnica requiere de un gran número de

configuraciones y varios parámetros resultan confusos para muchos usuarios. En la Figura 4 se muestra un ejemplo de este modelo.

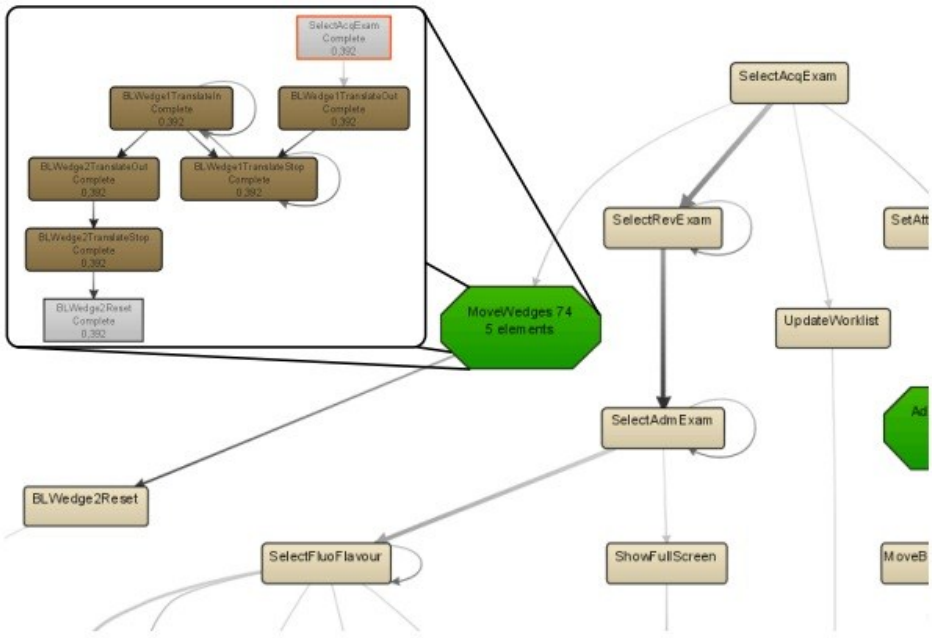


Figura 4: Modelo difuso (Günther, 2009)

1.3.4. Minería de variantes.

Esta técnica de diagnóstico de proceso (Figura 5) se basa en la identificación de variantes del modelo de procesos. Permite la detección jerárquica de subprocesos y los patrones de control de flujo asociados. Para cada variante identificada se genera un perfil de diagnóstico con información relevante del proceso (Alfonso, y otros, 2014).

La generación de variantes consiste en la construcción de variantes de modelos de proceso dadas por diferentes descomposiciones en subprocesos considerando los patrones de control de flujo (secuencia, lazo, paralelismo, selección exclusiva (XOR) y selección no exclusiva (OR)). Se utiliza un patrón de control de flujo para descomponer un subproceso descartando el comportamiento poco frecuente y/o asumiendo la presencia de comportamiento. Por tanto, dependiendo del comportamiento descartado y/o asumido es posible descomponer cada subprocesos mediantes diferentes patrones de control de flujo (Alfonso, y otros, 2014).

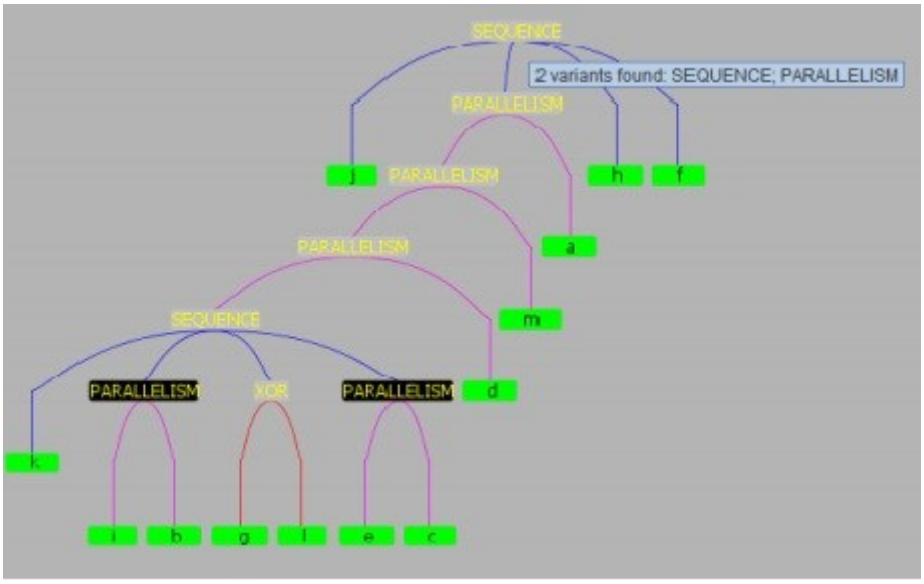


Figura 5: Árbol de Variantes (Alfonso, y otros, 2014)

Esta técnica muestra información relevante relacionada con el proceso y las características presentes en el registro de eventos. La obtención de un Árbol de Variantes posibilita enmarcar las actividades y las características identificadas en su contexto específico dentro del proceso. Sin embargo, al Árbol de Variantes, al ser un modelo jerárquico, no se le puede aplicar las métricas existentes para medir su calidad, porque en su gran mayoría se aplican a modelos obtenidos bajo la notación de redes de Petri.

1.4. Redes de Petri (del inglés Petri nets).

La red de Petri es un formalismo de modelado, lleva el nombre de Carl Adam Petri, quien ideó este lenguaje para apoyar el diseño y el análisis de procesos de negocio. Es ampliamente utilizada en el mundo académico por su expresividad y la concisión. No sólo es un lenguaje descriptivo que puede retratar a los procesos de negocio de una manera clara, de forma gráfica, sino que también se puede aplicar para llevar a cabo la simulación y análisis de procesos, lo que también se conoce como un modelo ejecutable (Ma, 2012).

1.4.1. Definición de una red de Petri.

Una red de Petri es un grafo dirigido bipartido formado por nodos lugares y nodos transiciones conectados por arcos y se define de la siguiente forma: Una red de Petri es un triplo N= (P, T, F) donde P es un conjunto finito de lugares, T es un conjunto finito de

transiciones tal que P ∩ T = ∅ y F ⊆ (P × T) ∪ (T × P) es un conjunto de arcos dirigidos, llamado relación de flujo (van der Aalst, 2011) (Aalst, 1995).La Figura 6 es un ejemplo de una red de Petri.

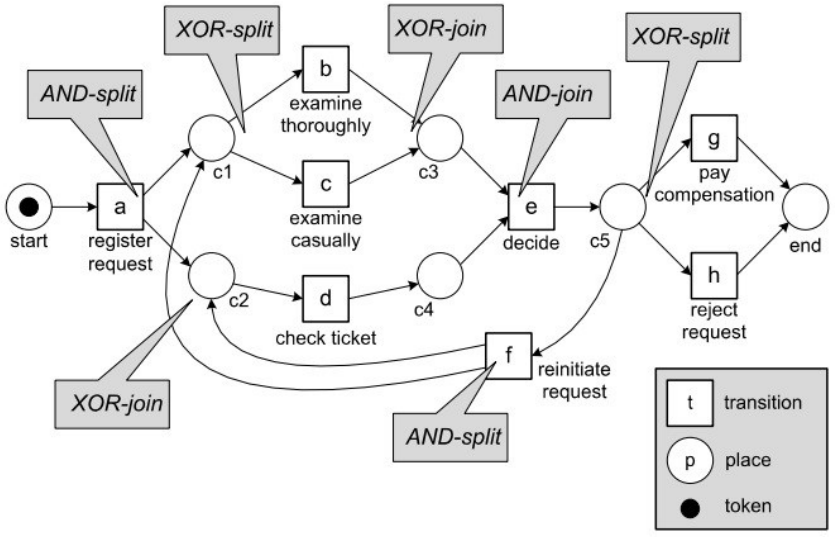


Figura 6: Ejemplo de los elementos y de las estructura de control básica de una red de Petri (van der Aalst, 2011)

Esta se puede expresar de la siguiente forma (van der Aalst, 2011): P={inicio ,c1,c2,c3,c4,c5, fin}, T={a, b, c, d, e, f, g, h}, and F={(inicio, a), (a, c1), (a, c2), (c1,b), (c1, c), (c2, d), (b, c3), (c, c3), (d, c4), (c3, e), (c4, e), (e, c5), (c5,f), (f, c1), (f, c2), (c5, g), (c5, h), (g, fin), (h, fin)}.

1.4.2. Elementos de una red de Petri.

Una red de Petri tiene como elementos: lugares, transiciones, los arcos que los relacionan y Tokens (ver Figura 6). Los lugares son elementos pasivos, se representan con círculos y pueden jugar diferentes roles (Aalst, 1994) (Ma, 2012):

- Un tipo de medio de comunicación, como una línea telefónica, un intermediario, o una red de comunicación.
- Un buffer: por ejemplo, un depósito, una cola.
- Una localización geográfica, como un lugar en un almacén, una oficina o un hospital.

- Un posible estado o condición de estado: por ejemplo, el piso donde se encuentra un elevador, o la condición de que un especialista esté accesible u ocupado.

Los Tokens pueden existir en los lugares y no pueden existir en las transiciones. En un momento dado, un lugar contiene cero o más Tokens, los cuales son representados por puntos negros. Su cantidad puede cambiar durante la ejecución de la red y estos pueden jugar los siguientes roles (Aalst, 1994) (Ma, 2012):

- Un objeto físico, por ejemplo un producto, una parte, una persona.
- Un objeto de información, por ejemplo un mensaje, una señal, un reporte.
- Una colección de objetos, por ejemplo un camión con productos, a almacén con partes, o un fichero de direcciones.
- Un indicador de un estado, por ejemplo el indicador del estado en el cual un proceso está o el estado de un objeto.
- Un indicador de una condición: la presencia de un Token indica cuando una cierta condición es satisfecha.

Las transiciones son elementos activos, representados por cuadrados y pueden jugar los siguientes roles (Aalst, 1994) (Ma, 2012):

- Un evento: por ejemplo, el inicio de una operación, la muerte de un paciente, un cambio de estación o la alternación de la luz del tráfico de rojo a verde.
- La transformación de un objeto: como adaptar un producto, actualizar una base de datos, o actualizar un documento.
- El transporte de un objeto: por ejemplo, transportar bienes, o enviar un archivo.

Por último, un arco es una conexión dirigida entre un lugar y una transición en una red de Petri. In El arco indica las relaciones entre los lugares y las transiciones, así como el flujo de trabajo del proceso.

Entradas y salidas de las transiciones.

Un lugar *p* se denomina entrada de una transición *t* si y solo si existe un arco dirigido desde el lugar *p* hasta la transición t. Un lugar *p* se denomina salida de una transición *t* si y solo si existe un arco dirigido desde la transición *t* hasta el lugar *p* (Aalst, 1995) (Aalst, 1994; van der Aalst, 2011).

Ejecución de las transiciones en una red de Petri (Disparo de una transición).

La ejecución o realización de una transición se denomina disparo (van der Aalst, 2011) (Aalst, 1995).

- Una transición para dispararse tiene que estar habilitada.
- Una transición t se dice que está habilitada si y solo si cada uno de los lugares de entrada p de t contiene al menos un Token.
- Cuando una transición t se dispara, la misma consume un Token de cada uno de sus lugares de entrada •t y produce un Token en cada uno de sus lugares de salida t•.

1.4.3. Estado o marca de una red de Petri.

Una red de Petri marcada es un par (N, M), donde N = (P, T, F) es una red de Petri y donde M ∈ B (P), o sea M: P→N que asocia al conjunto de lugares con números naturales (N incluye al 0) (van der Aalst, 2011). El estado M de la red se puede representar de la siguiente forma: M=n1p1+n2p2+… +NmPm. En la Figura 7 se muestra una red de Petri con estado: M=1p1+0p2+2p3+1p4.

La marca o estado de la red de Petri también se puede denotar como un arreglo.Por ejemplo el estado de la Figura 7 se denota de la siguiente manera: M= (1, 0, 2,1).

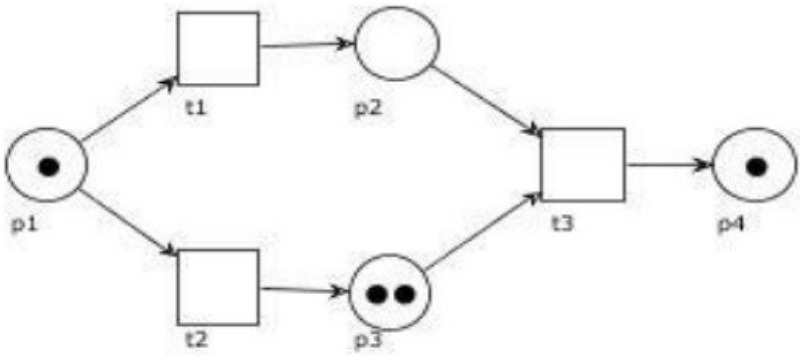


Figura 7: Red de Petri (Murata, 1989)

1.4.4. Estructuras de control típicas de las redes de Petri.

Las redes de Petri emplean las siguientes estructuras de control:

- Causalidad
- Paralelismo (AND-split - AND-join)
- Selección (XOR-split – XOR-join)
- Iteración (XOR-join - XOR-split)
- Restricciones de Capacidad (lazo de retroalimentación, exclusión recíproca, alternar).

1.5. Tecnologías, lenguajes y herramientas a utilizar.

Para el desarrollo de la solución fueron seleccionadas un conjunto de tecnologías, lenguajes y herramientas, que se describen a continuación.

1.5.1. Proceso Unificado Ágil (AUP por sus siglas en inglés).

Para el correcto desarrollo de software no es suficiente contar con notaciones de modelado y herramientas, sino que también es importante la dirección a seguir para la correcta aplicación de los demás elementos. La aplicación de una metodología de desarrollo es la que provee esta dirección.

AUP metodología formalizada por Scott W. Ambler que se preocupa especialmente de la gestión de riesgos. Al igual que en el Proceso Unificado de Desarrollo (RUP en inglés), en AUP se establecen cuatro fases que transcurren de manera consecutiva y que acaban con hitos claros alcanzados:

- **Inicio (Concepción):** El objetivo de esta fase es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.

- **Elaboración:** El objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.

- **Construcción:** Durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.

- **Transición**: El sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación y aceptación y finalmente se despliega en los sistemas de producción.

La metodología AUP establece un modelo más simple que el que aparece en RUP, por lo que reúne en una única disciplina las disciplinas de Modelado de Negocio, Requisitos y Análisis y Diseño. El resto de disciplinas (Implementación, Pruebas, Despliegue, Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP. Las disciplinas AUP quedan definidas de la siguiente forma (Ambler, 2014):

- **Modelado:** El objetivo de esta disciplina es entender el negocio de la organización, el dominio del problema se aborda en el proyecto, y e identificar una solución viable para hacer frente al dominio del problema.

- **Implementación**: Tiene como objetivo transformar su modelo en código ejecutable y realizar un nivel básico de las pruebas, en particular, las pruebas unitarias.

- **Pruebas**: Tiene como objetivo llevar a cabo una evaluación objetiva para garantizar la calidad. Esto incluye encontrar defectos, validar que el sistema funciona tal como fue diseñado y verificar que se cumplen los requisitos.

- **Despliegue:** Su finalidad es hacer planes para el suministro del sistema y para ejecutar el plan para que el sistema esté disponible para los usuarios finales.

- **Gestión de la configuración:** Tiene como propósito administrar el acceso a sus artefactos del proyecto. Esto incluye no sólo el seguimiento de versiones de artefactos en el tiempo, sino también el control y la gestión de los cambios a los mismos.

- **Gestión de proyecto:** Su intención es dirigir las actividades que lleva a cabo el proyecto. Esto incluye administrar los riesgos, dirigir a las personas (la asignación de tareas, seguimiento del progreso, etc.), y la coordinación con las personas y los sistemas fuera del alcance del proyecto para asegurarse de que está entregado a tiempo y dentro del presupuesto.

- **Entorno:** El objetivo de esta disciplina es apoyar el resto de los esfuerzos por garantizar que el proceso adecuado, orientación (normas y directrices) y herramientas (hardware, software, etc.) están disponibles para el equipo cuando sea necesario.

Esta metodología es la seleccionada para llevar a cabo el desarrollo de la aplicación, debido a que está basada en la metodología RUP estudiada en la carrera, y que como es una versión ágil de esta, permite generar los artefactos que se consideren necesarios, en dependencia del proyecto que la utilice. La racionalización de las fases en AUP reduce el tiempo de desarrollo y permite utilizar cualquier tipo de herramientas. Además, no necesita un gran número de desarrolladores por lo que se adecua a proyectos pequeños como este trabajo y es la metodología que se emplea en el grupo de investigación.

1.5.2. Lenguaje de Modelado Unificado.

El Lenguaje Unificado de Modelado (UML) es un lenguaje para especificar, visualizar construir y documentar los artefactos de los sistemas de software, así como para el modelado del negocio. Además de que permite la modelación de sistemas con tecnología orientada a objetos, es el lenguaje de modelado propuesto por la metodología AUP (Larman, 2005).

1.5.3. Java.

Se utiliza el lenguaje de programación Java, ya que es un lenguaje avanzado con el que se puede implementar cualquier tipo de programa, desde una aplicación web o desktop hasta aplicaciones para sistemas embebidos. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia en la informática. Es desarrollado por la empresa Sun Microsystems enfocado a cubrir las necesidades tecnológicas de punta (Arnow, y otros, 2001). Entre sus principales características destacan:

- Metodología de la programación orientada a objetos.
- Ejecución de un mismo programa en múltiples sistemas operativos.
- Fácil de usar y tomando lo mejor de otros lenguajes orientados a objetos, como C++.

1.5.4. Visual Paradigm.

Visual Paradigm es una herramienta CASE (Computer Aided Software Engineering traducido al español Ingeniería de Software Asistida por Computadora) que utilizando el lenguaje de modelado unificado (Unified Modeling Language, UML) como lenguaje de modelado, permite la captura, el diseño, la gestión y documentación de los artefactos generados durante el proceso de desarrollo de software. Además, es de código abierto, multiplataforma y consume pocos recursos (Paradigm, 2012).

Entre las principales características que condicionaron su selección para la investigación se encuentra la agilización a la hora de realizar los diagramas UML, ingeniería inversa de código a modelo y de código a diagrama; y la sincronización entre modelos y código. La versión que se empleará será la 8.0.

1.5.5. Eclipse.

La plataforma de desarrollo a usar es Eclipse, esto se debe a que cuenta con una amplia documentación y soporte en la comunidad de desarrolladores de Java. Este Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) es de código abierto y basado en Java, además de soportar otros lenguajes de programación como C/C++, Python, PHP, entre otros. Es soportado por varios sistemas operativos como Windows y GNU/Linux, para ello hace uso de la máquina virtual de Java (JVM). Este IDE presenta un potente editor con un buen completamiento de código que es muy bien recibido para el desarrollo del proyecto. Se empleará la versión Eclipse 4.2, la cual goza de gran estabilidad, soporte y en general se desarrolla en la JVM Java SE versión 6.0 (Eclipse, 2010).

1.5.6. ProM.

ProM es un marco extensible que soporta una amplia variedad de técnicas de Minería de Procesos en forma de complementos. Es independiente de la plataforma ya que se implementa en Java y se puede descargar de forma gratuita. ProM 6.2 se distribuye en partes, lo que ofrece una gran flexibilidad. Los complementos se pueden descargar e instalar sin restricciones (Verbeek, y otros, 2010). El marco de ProM recibe como registros de entrada registros de eventos en formatos XES o formato MXML que contienen los datos para la ejecución de las actividades de los sistemas. Debido a estas características

es la herramienta por excelencia para la Minería de Procesos y es la más utilizada por la comunidad internacional que investiga el área y la utilizada por el grupo de investigación de la Facultad 3.

1.5.7. Cobefra.

Es un sistema desarrollado en Java bajo la Licencia Pública General (GPL, del inglés General Public License), el cual tiene implementado toda una serie de métricas para la evaluación de modelos de procesos. Cobefra, cuyas métricas están divididas en las cuatro dimensiones de calidad, solo puede analizar los modelos de procesos basados en redes de Petri (Broucke, y otros, 2013).

1.6. Conclusiones parciales.

En el capítulo fueron estudiados varios conceptos referentes a la Minería de Procesos. Se realizó un estudio de las principales técnicas de diagnóstico de procesos y sus deficiencias, seleccionándose la minería de variantes para transformarla a redes de Petri, debido a que esta técnica mejora el trabajo con el ruido pero no se le pueden aplicar métricas porque no se encuentra en la notación red de Petri.

Se realizó un estudio de la metodología, las herramientas y el lenguaje a emplear. Se optó por la metodología AUP y se definieron los artefactos a generar que serán los siguientes: en la fase Negocio, Modelo dominio; en la fase Requisitos, Especificación de requisitos; en la fase Diseño, Diagrama de clases; en la fase de Implementación, Diagrama de componentes y Diagrama de despliegue. Como herramienta CASE se escogió Visual Paradigm por ser de código abierto y compatible con Java. Se escogió el IDE Eclipse y el lenguaje Java para el desarrollo de la aplicación, pues Java es el lenguaje más usado por la comunidad minera para el desarrollo de técnicas de minería de procesos. Se escogió ProM como marco de trabajo de Minería de Procesos pues está concebido para admitir la adición de complementos y de esta manera posibilitar el desarrollo de nuevos algoritmos y técnicas en el campo de la minería de proceso, además, es el marco de trabajo usado por los investigadores de esta rama. Cobefra es el marco de trabajo seleccionado para aplicar las pruebas de calidad del modelo de red de Petri obtenido, debido a las métricas que ya tiene implementadas.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN

2.1. Introducción.

En el presente capítulo se describe la propuesta de solución generando los artefactos propios de la metodología seleccionada. Se representa el negocio a través del modelo de dominio. Tomando este como punto de partida, se definen y especifican los requisitos funcionales y no funcionales que se deben tener en cuenta en el sistema. Se diseñan las clases a implementar. Además, se describen los diagramas de componentes y despliegue, los patrones y estándares a utilizar para el desarrollo del complemento.

2.2. Modelo de dominio.

El Modelo de dominio es la representación visual de los principales conceptos y sus relaciones, así como datos que son de interés para la investigación. La decisión de realizar un modelo de dominio se toma a partir de no tener una claridad suficiente en los procesos como para modelar un negocio (Jacobson, y otros, 2000). En la Figura 8 se muestra el modelo de dominio de la solución. Los principales conceptos identificados son:

Registro de eventos: Guarda información sobre las actividades y los estados de los procesos que se gestionan en las empresas.

Minería de Variantes: A partir de un registro de eventos construye variantes de modelos de procesos dadas por diferentes descomposiciones en subprocesos, considerando los patrones de control de flujo: secuencia, lazo, paralelismo, selección exclusiva y selección no exclusiva.

Conver Vtree To Petri Net: Devuelve un modelo en notación red de Petri, donde se representa los subprocesos y las actividades resultantes de la aplicación de la técnica Minería de Variantes. Este modelo puede ser empleado por otros complementos para ser análisado.

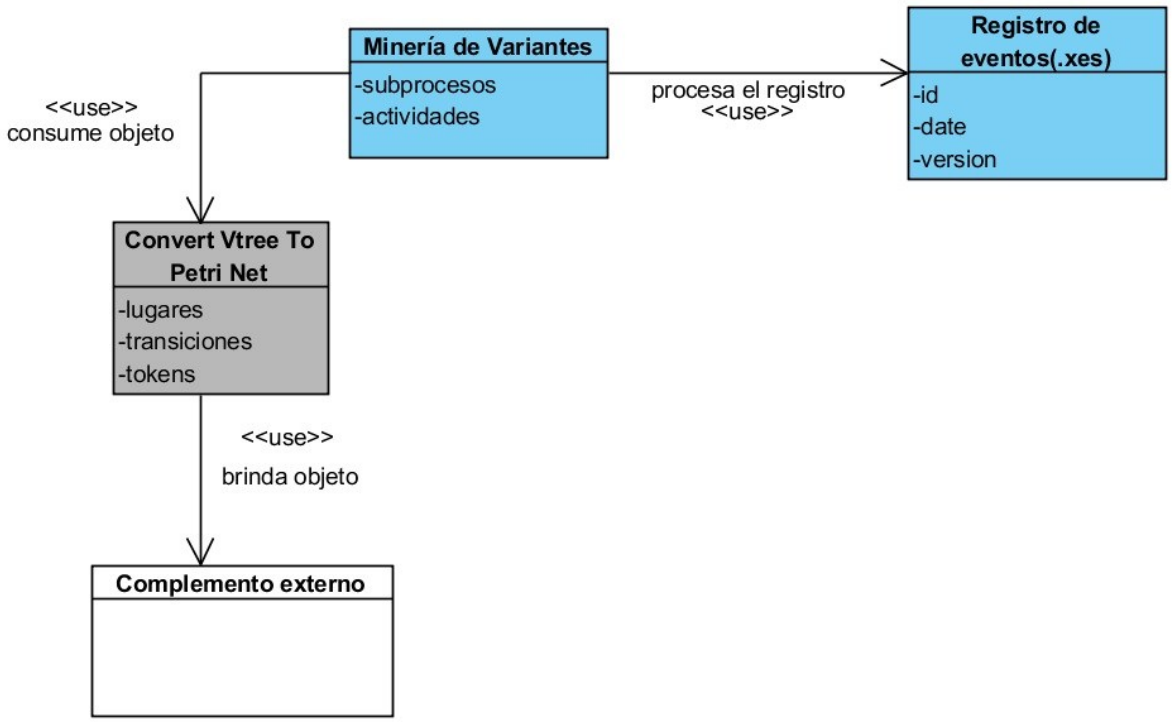


Figura 8: Modelo de dominio

2.3. Requisitos del software.

Los requisitos del software son las características que se deben exhibir por el software desarrollado o adatado, para solucionar un problema particular. Por lo tanto debe definir exactamente lo que se va a implementar (Sommerville, 2005) (Abran, y otros, 2004).

2.3.1. Técnicas para la captura de requisitos.

Para la obtención de los mismos se realizaron algunas técnicas de levantamiento de requisitos:

- **Tormentas de ideas:** Se realizaron reuniones de grupo de trabajo, donde los participantes pudieron exponer sus criterios sin ser interrumpidos, con el fin de obtener ideas más refinadas para la obtención de requisitos.
- **Arqueología de documentos:** Se consultaron diferentes bibliografías existentes sobre las técnicas de descubrimientos de procesos en la Minería de Procesos, permitiendo un mayor entendimiento del desempeño de estas técnicas, contribuyendo a la identificación de los requisitos.

- **Revisión de sistemas informáticos existentes:** Se analizaron varios complementos del ProM con el objetivo de identificar características y funcionalidades que pudieran ser incluidas en el sistema en desarrollo.

2.3.2. Requisitos funcionales.

Los requisitos se clasifican como requisitos funcionales y no funcionales. Los requisitos funcionales de un sistema describen con detalle lo que el sistema debe hacer, es decir, la función de este, sus entradas, salidas y excepciones (Sommerville, 2005).

A continuación se presentan los requisitos funcionales capturados y su especificación puede ser consultada en el documento “Especificación de requisitos” del expediente de la investigación.

RF.1 Transformar un modelo de Árbol de Variantes en una red de Petri.

RF.2 Reubicar los nodos en el modelo.

RF.3 Agrandar el modelo.

RF.4 Exportar la red de Petri visualizada.

RF.5 Importar el árbol.

2.3.3. Requisitos no funcionales.

Los requisitos no funcionales son aquellos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento, por mencionar algunas (Sommerville, 2005).

Los requisitos no funcionales definidos en el presente trabajo se muestran a continuación:

Hardware:

- Se necesita como mínimo un procesador Intel Pentium IV con CPU 3.0 GHz.
- Se necesita como mínimo 1 GB de RAM.
- Se necesita como mínimo una capacidad de almacenamiento de 5 GB libres.

Software:

- Se necesita como Sistema Operativo Windows XP o superior, o cualquier distribución de Linux con kernel 2.3.6 o superior.
- Se necesita Máquina Virtual de Java en su versión 7.
- Se necesita ProM en su versión 6.2. y el paquete VariantMiner (Minería de Variantes).

Usabilidad:

- La interfaz debe ser amigable, para que el usuario por intuición aprenda a trabajar en ella.

Integración:

- El modelo obtenido debe poder ser analizable por otros complementos del marco de trabajo ProM.

Portabilidad:

- El sistema debe ser multiplataforma, es decir, se podrá usar en sistemas Windows o Linux y para poder usarse en cualquier estación de trabajo se debe convertir en un complemento para ProM.

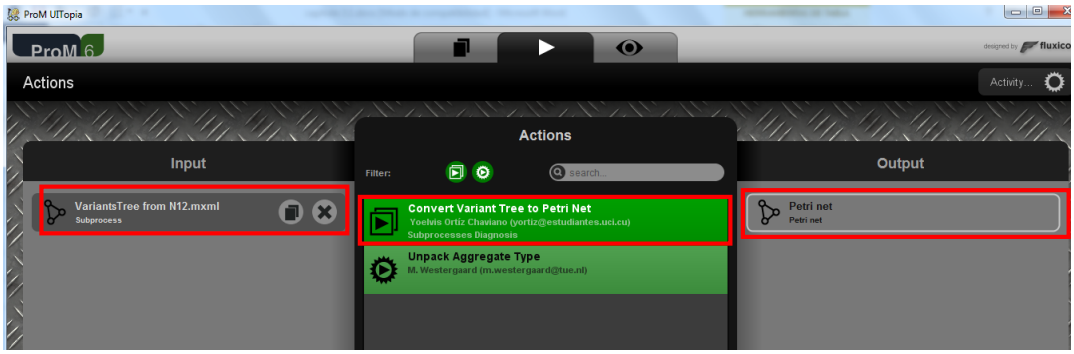
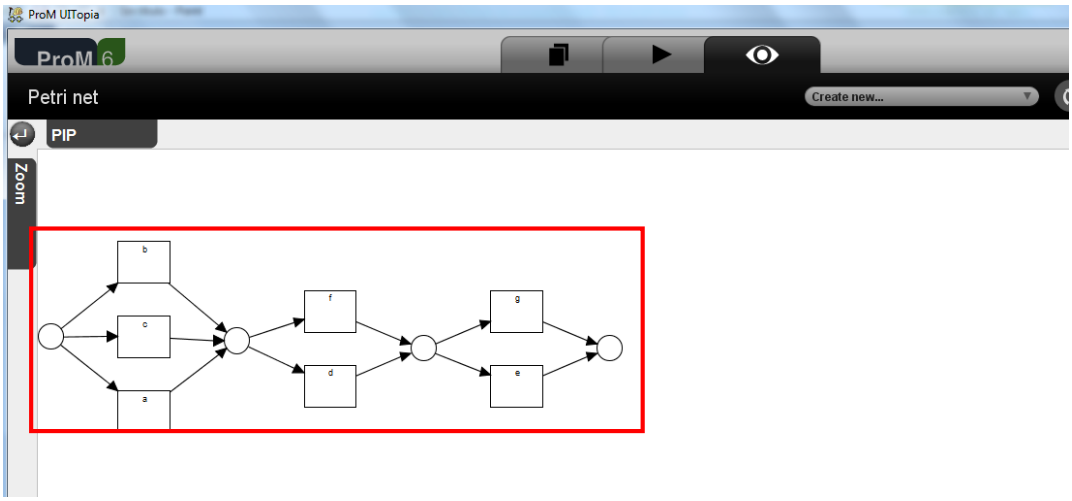
Implementación:

- El sistema debe ser programado en el lenguaje Java para que coincida con el lenguaje en el que se desarrolló ProM.

2.3.4. Especificación de requisitos.

La especificación de requisitos es un documento escrito, combinado con descripciones en lenguaje natural, imágenes y modelos gráficos del comportamiento del sistema que se va a desarrollar (Pressman, 2005).

A continuación se presenta en la Tabla 1 la especificación del requisito Transformar un modelo de Árbol de Variantes en una red de Petri:

Nº	Nombre	Descripción	Complejidad	Prioridad para cliente
RF1	**RF.1** Transformar un modelo de Árbol de Variantes en una red de Petri.	Convierte un Árbol de Variantes en una red de Petri	*Alta*	*Alta*
	Prototipo			
				
				

Campos	Tipos de Datos	Reglas o Restricciones
Search	*Cadena de caracteres*	Opcional
Tipo	*Nomenclador*	Opcional
Observaciones		

Tabla 1: Especificación del requisito RF.1

La complejidad de los requisitos se determinó utilizando una técnica de estimación propuesta por el Programa de Mejora del Centro CEGEL, donde se definen un grupo de indicadores tales como: complejidad por interfaces, formas de inicialización, restricciones de validación, grado de reutilización y lógica del negocio.

A continuación se muestra la cantidad de requisitos que presentan baja, media y alta complejidad:

- 2 de complejidad baja.
- 2 de complejidad media.
- 1 de complejidad alta.

2.3.5. Validación de requisitos.

La validación de requisitos se encarga de comprobar que estos realmente definen el sistema que se quiere hacer. Recurrir a estas acciones es importante debido a que los errores en la especificación de los requisitos pueden inducir a importantes costos al repetir el trabajo cuando son descubiertos durante el desarrollo o después de que el sistema esté en uso (Sommerville, 2005).

Para la validación de requisitos se utilizó la técnica de revisión de requerimientos la cual consiste en inspeccionar o hacer revisiones del documento de especificación de requisitos con la meta de encontrar errores, asunciones confundidas y la carencia de claridad (Abran, y otros, 2004). Además de esta se emplearon otras tres técnicas: Estabilidad de requisitos, Grado de validación de los requisitos y Especificidad de requisitos, que se muestran a continuación:

- **Estabilidad de requisitos:**

Después de realizar un análisis con el cliente se identificaron 6 requisitos funcionales, de los cuales uno sufrió modificaciones. Luego se aplica la formula siguiente:

$$\boldsymbol{ETC = \frac{[RT - RM] * 100}{RT}}$$

$$\boldsymbol{ETC = \frac{[5 - 1] * 100}{5} = 80}$$

Donde:

***ERT*:** Estabilidad de los requisitos.

RT: Total de requisitos definidos.

RM: Cantidad de requisitos modificados

La técnica arrojo como resultado un 80 % de estabilidad en los requisitos, por lo que se asegura que los requisitos permanecerán inalterables. Por consiguiente es posible trabajar en el análisis y diseño sobre ellos sin futuros contratiempos.

- **Grado de validación de requisitos:**

Esta técnica se emplea para medir que los requisitos identificados con el cliente son los correctos.

$$\boldsymbol{VR = \frac{NC}{(NC + NNV)}}$$

$$\boldsymbol{VR = \frac{5}{(5 + 0)} = 1}$$

Donde:

VR: Grado de validación de requisitos

NC: Número de requisitos que han sido validados como correctos

NNV: Número de requisitos no validados aún.

A partir del resultado obtenido con esta técnica se evidencia que existe un alto nivel de corrección en la definición de los requisitos.

- **Especificación de requisitos:**

Con esta técnica se mide el grado de ambigüedad de los requisitos

$$Q1 = \frac{NUI}{NR}$$

$$Q1 = \frac{5}{5} = 1$$

A partir de los resultados de esta técnica se evidencia que no existe ambigüedad en los requisitos, por lo que la especificación de los requisitos se hizo correctamente.

2.4. Diseño de la solución.

En el presente epígrafe se muestran la descripción de los diferentes diagramas de clases realizados en el diseño así como el empleo de los patrones GRASP, los patrones GoF y las métricas para validar el diseño.

2.4.1. Diagrama de paquetes.

El Diagrama de paquetes organiza las clases en paquetes cuando el diagrama de clases está muy cargados de clases, para una mejor comprensión durante el desarrollo del software. En la Figura 9 se muestra la estructura de paquetes conformada a partir del diagrama de clases.

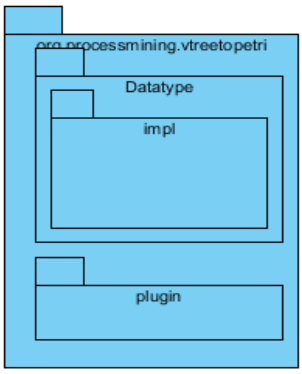


Figura 9: Diagrama de paquetes

2.4.2. Diagrama de clases.

En el diagrama de clases se visualiza las clases utilizadas en el diseño, sus atributos, asociaciones, la navegabilidad y dependencias. En la Figura 10 se muestran las clases del paquete impl de la aplicación.

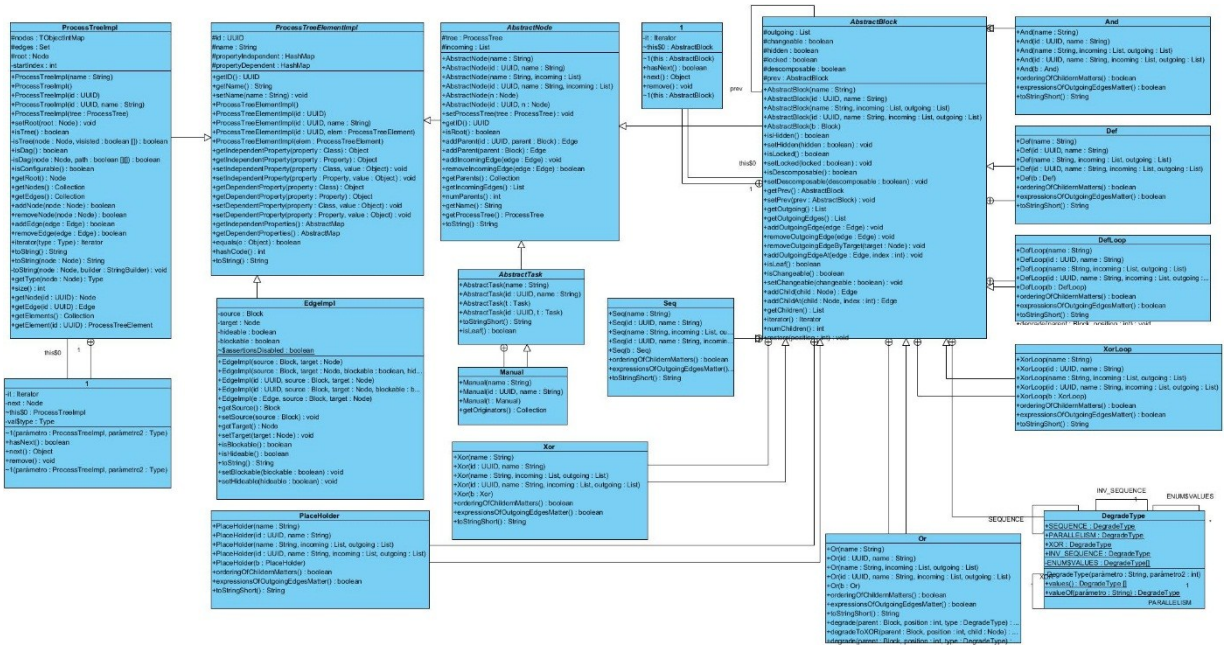


Figura 10: Clases del paquete impl

2.4.3. Patrones de diseño.

Los patrones de diseño son descripciones de clases cuyas instancias colaboran entre sí, que se adaptan para resolver un determinado tipo de problema en un contexto particular (Larman, 2005). Se utilizaron los siguientes:

Patrones GRASP: Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Dentro de este grupo de patrones se pueden encontrar los siguientes: Experto, Creador, Bajo Acoplamiento, Alta Cohesión y Controlador (Larman, 2005). Los patrones GRASP usados en la solución son los siguientes:

- **Experto:** Es usado cuando se necesitan clases con responsabilidades específicas de acuerdo a la información que gestionan. El empleo de este patrón se evidencia

por ejemplo (Figura 11) en las clases ConvertProcessTree y ConvertSubprocess, que tienen funciones específicas de acuerdo a la información que gestionan.

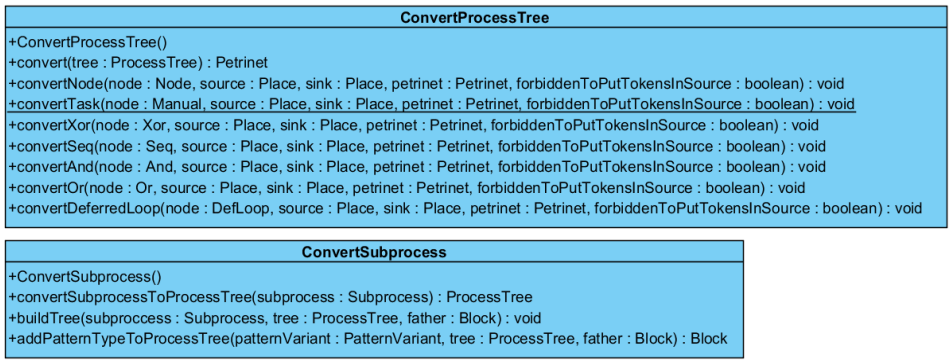


Figura 11 Ejemplo del patrón experto

- **Bajo acoplamiento:** Propone un diseño en el cual las clases son más independientes, propiciando la reducción de fallos al ocurrir cambios pues no dependen de otras clases para realizar sus funciones. Se emplea en las clases VtreeToPetriPlugin y en la mayoría de las clases del sistema, pues se relacionan a través de las interfaces que proveen las librerías de ProM y no con otras clases.
- **Alta cohesión:** La cohesión es una medida de cuan relacionada y orientadas están las responsabilidades de una clase. Un diseño con alta cohesión está caracterizado por tener clases con responsabilidades estrechamente relacionadas que no realizan tareas enormes. Ejemplo del uso de este patrón se evidencia en las clases VtreeToPetriExportPlugin y VtreeToPetriImportPlugin (Figura 12).
- **Controlador:** Plantea el uso de clases controladoras encargadas de manejar las peticiones del sistema. Este patrón se evidencia en la clase VtreeToPetriManager (Figura 12).

En la Figura 12 se evidencia el uso de estos patrones anteriormente mencionados.

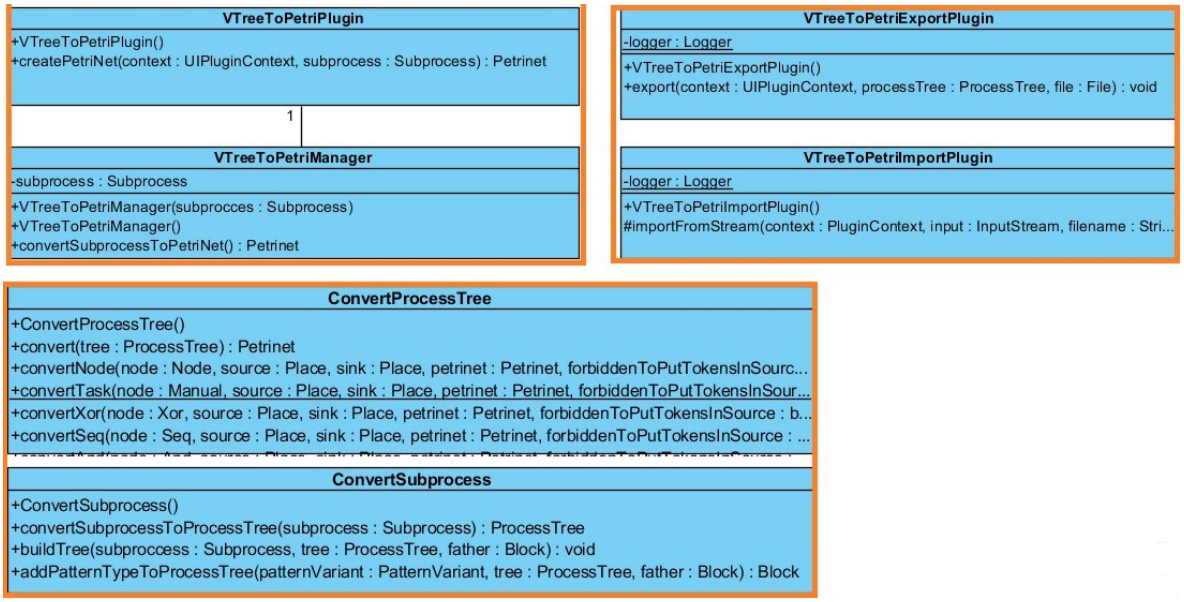


Figura 12: Clases del paquete plugin

Patrones GoF: Son un conjunto de patrones propuestos por el grupo Pandilla de Cuatro (Gang of Four en inglés) compuesto por los autores Erich Gamma, Richard Helm, Ralp Johnson y John Vlissides. Ellos proponen los patrones de diseño como un nuevo mecanismo para expresar la experiencia de diseño orientado a objetos (Gamma, y otros, 2002).

Los patrones GoF se dividen en tres categorías: creacionales, estructurales y de comportamiento. Los patrones creacionales son los que se refieren al proceso de creación de objetos, los estructurales se refieren a la composición de las clases u objetos y por último, los de comportamiento se caracterizan por las formas en que las clases u objetos interactúan y distribuyen responsabilidad. En la aplicación se utilizó un patrón de tipo estructural, que se muestra a continuación:

- **Composición:** Trata los objetos compuestos como si se tratase de uno simple. Se utiliza cuando se pretende usar una jerarquía recursiva de objetos. Cada componente en una interfaz jerárquica implementa la misma interfaz o hereda de una superclase común. En el desarrollo de la aplicación se usó en la clase ProcessTreeImpl (Figura 13).

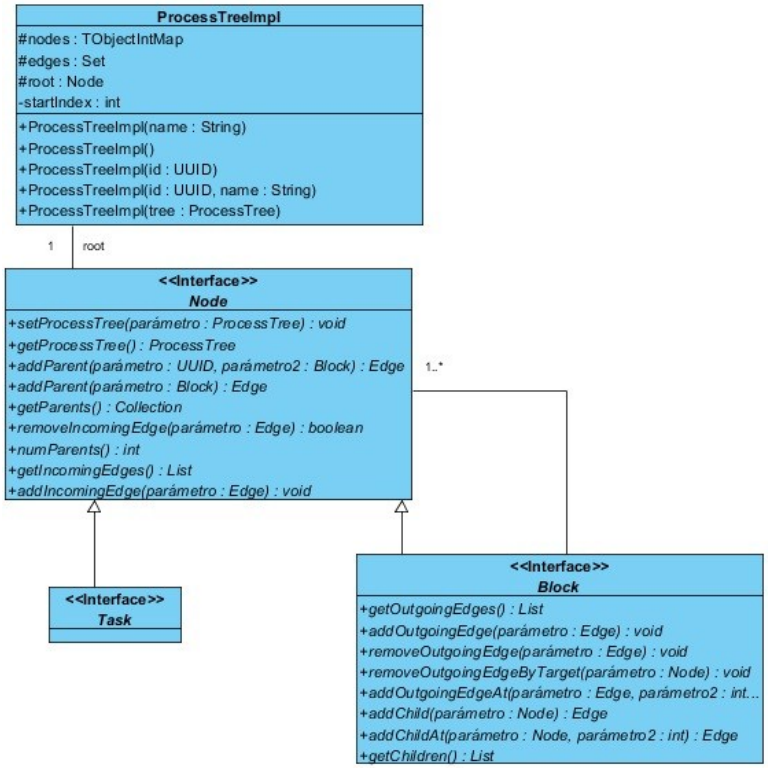


Figura 13 Ejemplo de patrón composición

2.4.4. Métricas de validación del diseño.

Las métricas del software proporcionan una manera cuantitativa de evaluar la calidad de los atributos internos de un producto y aseguran que los métodos no generan efectos colaterales dependiendo del estado. Con el empleo de métricas del software también se puede averiguar qué tan bien se definieron las clases y el sistema en general. De forma general, se garantiza el entendimiento y la mejora de la calidad del producto (Pressman, 2005). Las métricas utilizadas para evaluar la calidad del diseño propuesto son Tamaño Operacional de Clases (TOC) y Relaciones entre Clases, para evaluar la calidad del diseño propuesto.

2.4.5. Métrica Tamaño Operacional de Clase.

La métrica Tamaño Operacional de Clase (TOC en inglés) mide los siguientes atributos de calidad:

- **Responsabilidad:** Responsabilidad que posee una clase en un marco de modelado de un dominio correspondiente al modelado de la solución propuesta.

- **Complejidad de implementación:** Grado o dificultad que posee implementar un diseño de clases.
- **Reutilización:** Mide el nivel de reutilización de una clase o estructura dentro de un diseño de software.

A continuación se muestran en la Tabla 2 los criterios de evaluación de la métrica TOC.

	Categoría	Criterio
Responsabilidad	Baja	< =Promedio.
	Media	Entre Promedio. y 2* Promedio.
	Alta	> 2* Promedio.
Complejidad implementación	Baja	< =Promedio.
	Media	Entre Promedio. y 2* Promedio.
	Alta	> 2* Promedio.
Reutilización	Baja	> 2*Promedio.
	Media	Entre Promedio. y 2* Promedio.
	Alta	<= Promedio.

Tabla 2: Criterios de evaluación de los atributos para la métrica TOC

A continuación se presentan en la Figuras 14, 15 y 16 los resultados de la métrica TOC:

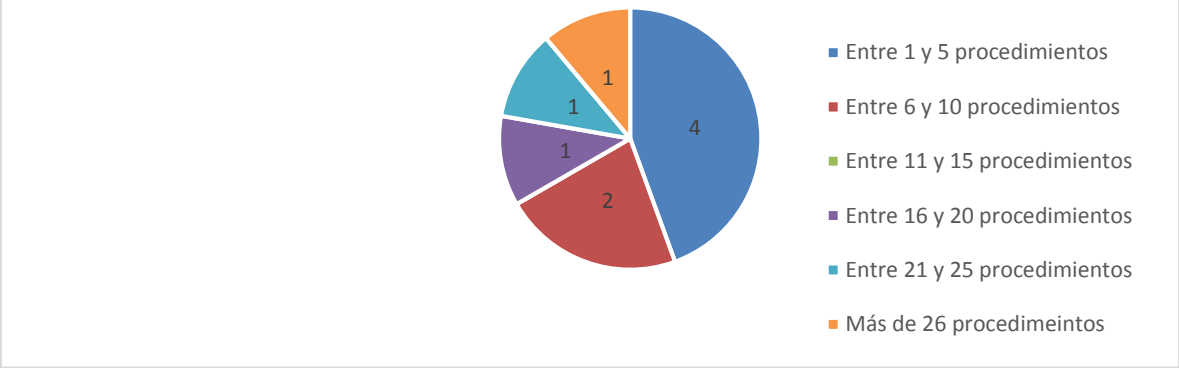


Figura 14: Número de clases por cantidad de procedimientos

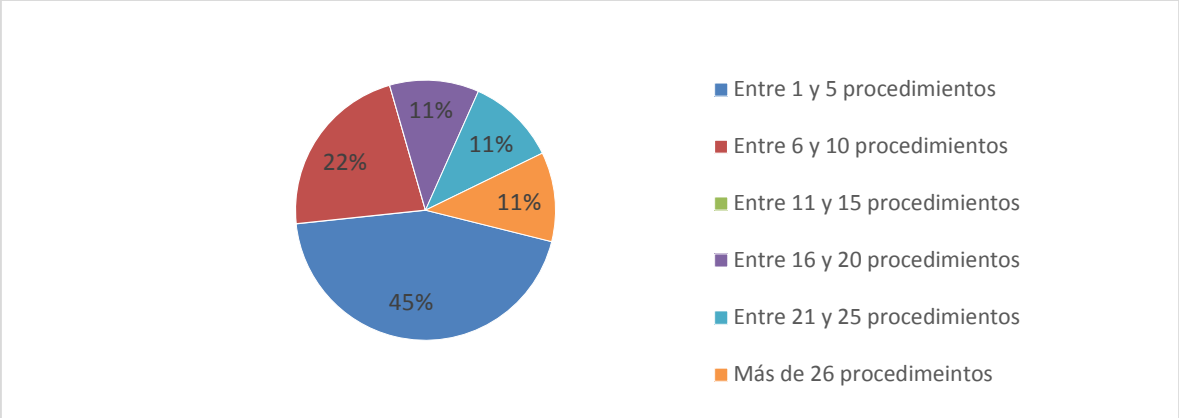


Figura 15: Distribución en porciento de las clases por cantidad de procedimientos

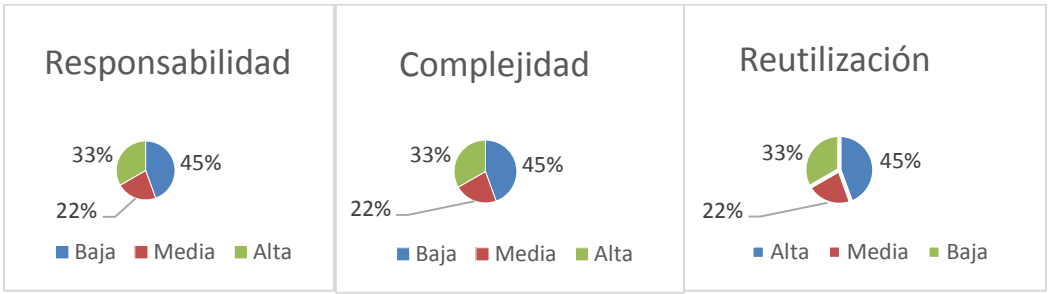


Figura 16: Resultados en porciento de las clases agrupadas según su responsabilidad, complejidad y reutilización

Interpretación de los resultados de la métrica Tamaño Operacional de Clases:

Los resultados que proporcionados por la métrica TOC se evidencia que existe un elevado por ciento de Reutilización. Además, se comprende que la Responsabilidad y Complejidad en las clases son bajas. Los resultados demuestran que los atributos de calidad se encuentran equilibrados con el predominio de baja responsabilidad y de baja complejidad, y de una alta reutilización. La alta capacidad de reutilización y la menor complejidad posible de las clases son elementos importantes en el desarrollo de software, y debido a los resultados mostrados, los atributos de calidad de las clases se encuentran a un nivel adecuado.

2.4.6. Métrica Relaciones entre Clases.

- **Acoplamiento:** Dependencia o interconexión de una clase o estructura de clase con otras.

- **Complejidad de mantenimiento:** Predice información crítica acerca de la confiabilidad y facilidad de mantenimiento del sistema propuesto.
- **Reutilización:** Mide el nivel de reutilización de una clase o estructura dentro de un diseño de software.
- **Cantidad de Pruebas:** Número o grado de esfuerzo necesario para realizar las pruebas de calidad al producto diseñado.

En la Tabla 3 se muestran los criterios de evaluación de los atributos para la métrica RC.

Atributo	Categoría	Criterio
Responsabilidad	Ninguna	0
	Baja	1
	Media	2
	Alta	> 2
Complejidad mantenimiento	Baja	< =Promedio.
	Media	Entre Promedio. y 2* Promedio.
	Alta	> 2* Promedio.
Reutilización	Baja	> 2*Promedio.
	Media	Entre Promedio. y 2* Promedio.
	Alta	<= Promedio.
Cantidad de pruebas	Baja	<= Promedio.
	Media	Entre Promedio. y 2* Promedio.
	Alta	> 2*Promedio.

Tabla 3: Criterios de evaluación de los atributos para la métrica RC

A continuación se presentan los resultados obtenidos en la métrica RC desde la Figura 17 hasta la Figura 20.

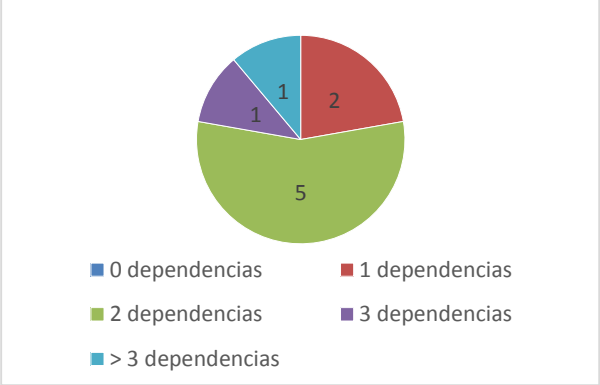


Figura 17: Número de clases por cantidad de dependencias

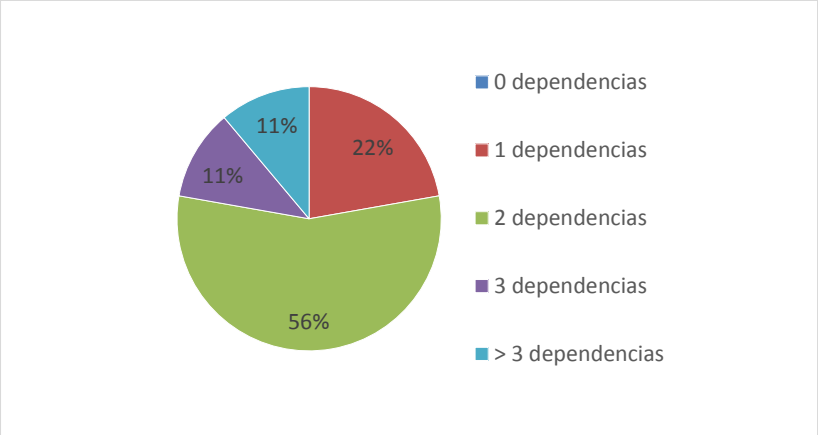


Figura 18: Agrupación de las clases en porciento por el número de dependencia que poseen.

Figura 19: Resultados en porciento de las clases agrupadas según su acoplamiento y complejidad de mantenimiento.

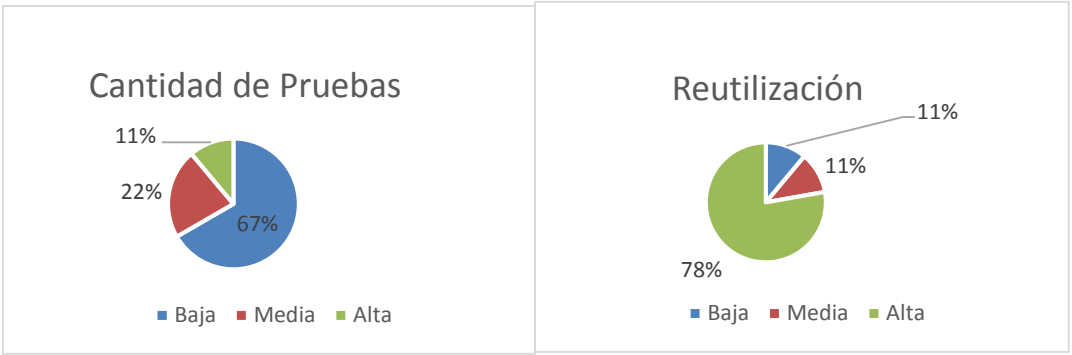


Figura 20: Resultados en porciento de las clases agrupadas según su reutilización y la cantidad de pruebas.

La interpretación de los resultados arrojados por la métrica RC es la siguiente:

Los resultados mostrados en las Figuras 17, 18, 19 y 20 muestran que el alto acoplamiento no sobrepasa el 22%. Tomando lo anterior y siendo 3 el número máximo de

dependencia de otras clases se garantiza la menor dependencia posible entre las clases del sistema, lo que provee un 56 % y un 22% de medio y bajo acoplamiento respectivamente. La cantidad baja de pruebas de un 67 % y alta reutilización de un 78 % demuestra que se favorece la reutilización de las clases y la implantación del diseño. Por tanto se llega a la conclusión de que el diseño presenta buena calidad.

2.5. Implementación.

La implementación utiliza las salidas del diseño y proporciona una de las entradas para las pruebas. Se desarrolla el software en términos de componentes, es decir, ficheros de código fuentes e interfaces, con el objetivo de implementar los artefactos definidos en el diseño (Abran, y otros, 2004).

2.5.1. Diagrama de componentes.

El Diagrama de componentes muestra los componentes, interfaces provistas y requeridas, puertos, y las relaciones entre ellos. (Fakhroutdinov, 2014). A continuación se muestra en la Figura 21 el diagrama de componentes utilizado en la investigación.

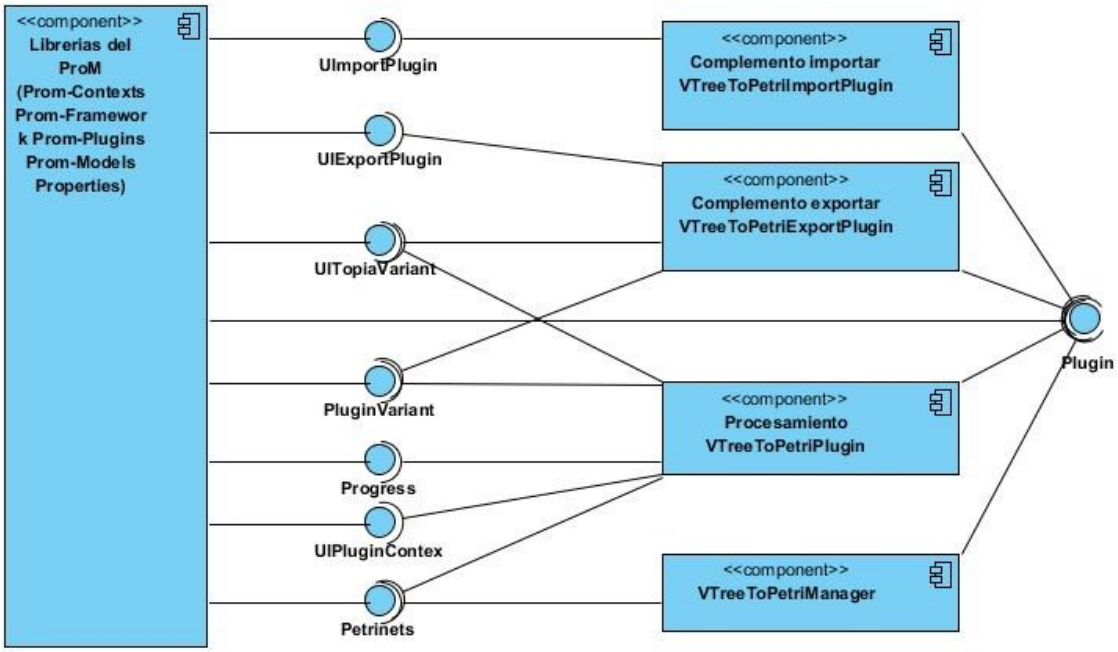


Figura 21: Diagrama de componentes.

2.5.2. Diagrama de despliegue.

El Diagrama de despliegue es un tipo de diagrama UML que se utiliza para modelar el hardware de los artefactos del software en nodos (Pressman, 2005). A continuación, en la Figura 22 se presenta el diagrama de despliegue de la investigación. En este componente se encuentra el registro de eventos y el marco de trabajo donde se desarrolla todo el proceso, dejando entrever que solo es necesario una computadora para el funcionamiento del sistema.

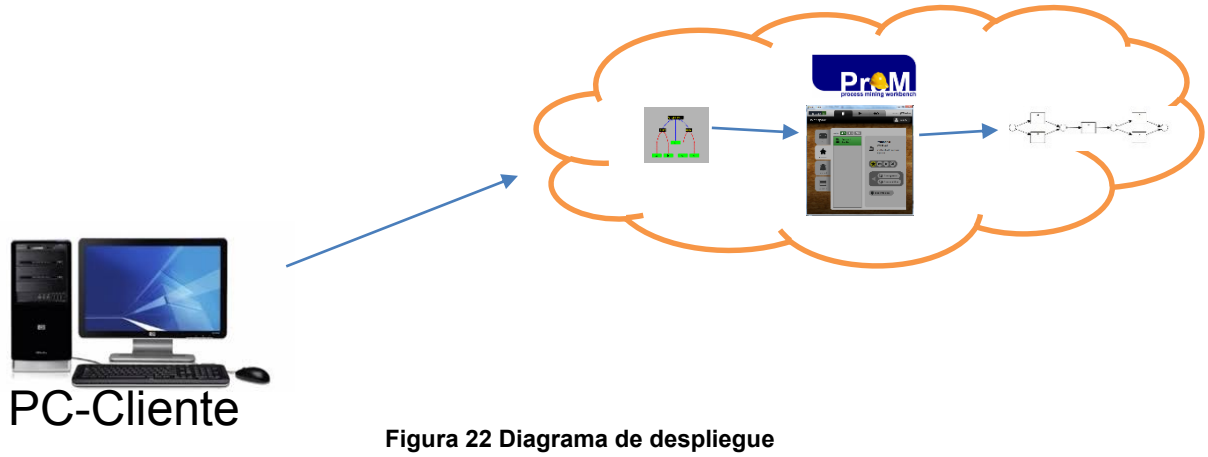


Figura 22 Diagrama de despliegue

2.5.3. Estándares de codificación.

Los estándares de codificación describen convenciones para escribir código fuente en lenguajes de programación. De esta forma se logra una limpieza y claridad en él código haciéndolo más legible. El estándar de codificación utilizado es el correspondiente al lenguaje Java. Este estándar viene detallado en el documento Convenciones de Código para el Lenguaje de Programación Java (Java Code Conventions en inglés). A continuación se muestran algunas de estas convenciones, en este caso, las convenciones de nombres utilizadas en la investigación (Hommel, 1999):

- Los métodos deben ser verbos; cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.
- Los nombres de las clases deben ser sustantivos; cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivas.

- Los nombres de las interfaces siguen la misma regla que las clases.
- Los nombres de las variables deben ser cortos pero con significado. La elección del nombre de una variable debe ser un mnemónico, designado para indicar a un observador casual su función. Los nombres de variables de un solo carácter se deben evitar, excepto para variables índices temporales. Nombres comunes para variables temporales son *i*, *j*, *k*, *m*, y *n* para enteros; *c*, *d*, y *e* para caracteres.

2.6. Conclusiones parciales.

Siguiendo la metodología AUP se generaron los artefactos necesarios para guiar y describir el desarrollo de la investigación. En el modelo de dominio se representaron los principales conceptos y sus relaciones pertenecientes al negocio. Se aplicaron técnicas de captura de requisitos como tormentas de ideas, análisis de documentos y de software similares logrando la correcta identificación y descripción de los requisitos. Para una mejor organización y comprensión se estructuraron en paquetes las clases utilizadas en el diseño, de las cuales se presentó una vista general de estas, sus atributos y métodos a través del diagrama de clases. Además, se aplicaron patrones GRASP y GoF para lograr un diseño correcto y evitar problemas comunes en el desarrollo de software. Se aplicaron las métricas TOC y RC evidenciando la existencia de un buen diseño de clases y adecuadas relaciones entre ellas. Durante la implementación el sistema fue divido en cinco componentes relacionados por interfaces, teniéndose en cuenta los estándares de codificación definidos por el libro “Convenciones de Código para el Lenguaje de Programación Java”.

CAPÍTULO 3: VALIDACIÓN

3.1. Introducción.

En el presente capítulo se realiza una descripción de las pruebas de Caja blanca y de funcionamiento que se aplicaron al complemento, con el fin de evaluar la calidad y comprobar sus funcionalidades, verificando en todos los casos que los resultados sean los esperados. Además, con intenciones de evaluar el modelo de variantes original, se describen varias métricas de minería de proceso que fueron aplicadas al modelo en notación de red de Petri obtenido con el complemento.

3.2. Pruebas de software.

Las pruebas del software son un proceso que intenta proporcionar confianza en un software y solo pueden demostrar la presencia de errores, no su ausencia. Las pruebas tienen el objetivo de evaluar y mejorar la calidad del producto, identificando defectos y problemas. Estas se aplican en diferentes fases evaluando el diseño y el cumplimiento satisfactorio de los requisitos (Sommerville, 2005) (Abran, y otros, 2004).

Para la validación de la calidad del complemento se emplean pruebas de caja blanca para comprobar el código del software en diversos puntos, es decir, la ejecución correcta de los métodos. Además, se realizan pruebas de caja negra para probar las funcionalidades y determinar si operan correctamente.

3.2.1. Pruebas de caja blanca.

Las pruebas de caja blanca se basan en el minucioso examen de los detalles procedimentales, obteniendo como resultado la detección de errores y por ende el aumento de la calidad y la fiabilidad del sistema (Pressman, 2005).

Las pruebas de caja blanca se pueden realizar de dos formas: estática y dinámica. En la forma estática se revisa el diseño del software, la arquitectura o el código sin ejecutarla. Algunas veces también se refiere a un análisis estructural. En la forma dinámica se basa sobre la prueba de un programa en ejecución donde se usa la información que obtiene de ver lo que hace el código y su funcionamiento para determinar qué vamos a probar, lo que

no debe probar, y la forma de abordar la prueba (Patton, 2006). Las pruebas de caja blanca dinámicas fueron las seleccionadas y se utilizó el marco de trabajo Junit para su realización.

Junit:

Se emplea Junit en su versión 4.5, el cual permite implementar pruebas en proyectos javas, ofrece un conjunto de librerías que se integra con varios IDEs, en este caso Eclipse. Además, permite realizar pruebas de auto verificación en Java y también proporciona clases que se utilizan en la creación de los casos de pruebas. También posee una interfaz gráfica simple donde se brinda la información del estado de las pruebas realizadas, es decir, si fallaron, dieron errores o sucedieron satisfactoriamente. (Rainsberger, 2005).

Entre las principales características se encuentran (Rainsberger, 2005):

- Métodos como el **setUp()** que permiten inicializar el entorno en que se ejecutarán las pruebas.
- Las pruebas se definen a través de un **TestCase** con métodos **testXXX()**.
- Para comprobar los resultados de las pruebas, previstos y reales, se invocan variantes de métodos **assertXXX()**.

Para aplicar las pruebas con Junit fue necesario realizar los siguientes pasos:

- Crear un caso de prueba por cada clase implementada.
- Definir dentro de la funcionalidad **setUp()** de Junit los tipos de datos de entrada para las pruebas.
- Definir los métodos que se probaran en cada caso de prueba.
- Proceder con las pruebas a los métodos.

Junit se utilizó durante toda la fase de implementación para evaluar si los métodos mostraban un correcto funcionamiento permitiendo la detección y corrección temprana de errores. A continuación se mostrará en las siguientes figuras un ejemplo de las pruebas de caja blanca realizadas a la clase **ConvertProcessTreeTest**. En la Figura 23 se puede ver el método **convert()** de la clase **CovertProcessTree** a la cual se le aplicarán las pruebas.

```
61    public Petrinet convert(ProcessTree tree)
62            throws NotYetImplementedException, InvalidProcessTreeException {
63        Petrinet petrinet = new PetrinetImpl(tree.getName());
64        Place source = petrinet.addPlace("source");
65        Place sink = petrinet.addPlace("sink");
66
67        convertNode(tree.getRoot(), source, sink, petrinet, false);
68        return petrinet;
69    }
```

Figura 23: Método conver() de la clase ConvertProcessTree

Se creó a continuación una clase de prueba llamada **ConvertProcessTreeTest**, en el paquete que contiene la clase original (Figura 24 y Figura 25), en la que se declaran los tipos de datos necesarios para probar cada método, en este caso el método **convert()**, el cual se encarga de convertir un árbol de procesos en una red de Petri. En la Figura 26 se muestra un ejemplo de una prueba realizada con la funcionalidad **assertNotNull()** al método **conver()** para determinar que este nunca devuelva ningún valor nulo.

VariantsTreeToPetriNetPlugin ▸ src ▸ org.processmining.vtreetopetri ▸ VTreeToPetri

```
17  public class ConvertProcessTreeTest {
18      ProcessTree tree;
19
20      @Before
21      public void setUp() throws Exception {
22          //ProcessTree de prueba
23          //Se utilizará de entrada
24          //en el método de Convertir de ProcessTree a Petri Net
25          this.tree = this.generarProcessTree();
26      }
27
28      /*
29       * Generar ProcessTree de prueba
30       */
31      public ProcessTree generarProcessTree() {
32          ProcessTree tree = new ProcessTreeImpl();
33          Edge edge;
34
35          // And( A , Seq (F,Z) )
36
37          Block and = new AbstractBlock.And("And");
38          and.setProcessTree(tree);
39          tree.addNode(and);
40          tree.setRoot(and);
41
42          Task a = new AbstractTask.Manual("A");
43          a.setProcessTree(tree);
44          tree.addNode(a);
45
46          edge = and.addChild(a);
47          a.addIncomingEdge(edge);
48          tree.addEdge(edge);
49
```

Figura 24: Clase ConvertProcessTreeTest

```
50          Block seq = new AbstractBlock.Seq("Seq");
51          seq.setDescomposable(true);
52          seq.setProcessTree(tree);
53          tree.addNode(seq);
54
55          edge = and.addChild(seq);
56          seq.addIncomingEdge(edge);
57          tree.addEdge(edge);
58
59          Task f = new AbstractTask.Manual("F");
60          f.setProcessTree(tree);
61          tree.addNode(f);
62
63          edge = seq.addChild(f);
64          f.addIncomingEdge(edge);
65          tree.addEdge(edge);
66
67          Task tau = new AbstractTask.Manual("Z");
68          tau.setProcessTree(tree);
69          tree.addNode(tau);
70
71          return tree;
72      }
```

Figura 25: Clase ConvertProcessTree.

```
    /*
     * En esta prueba de se verifica que el mÃ©todo convert
     * no devuelva nunca un valor Null (se verifica con assertNotNull de JUNIT)
     * MÃ©todo convert : Este mÃ©todo es el encargado de convertir de
     * un Process Tree a una Petri Net
     * El objeto tree es creado en el mÃ©todo setUp()
     *
     */
    @Test
    public void testConvert() throws NotYetImplementedException, InvalidProcessTreeException {
        ConvertProcessTree convertidor = new ConvertProcessTree();
        assertNotNull(convertidor.convert(tree));
    }
}
```

Figura 26: Prueba al método conver().

En la Figura 27 se muestran los resultados de las pruebas realizadas a la clase **ConvertProcessTree**, mostrando resultados satisfactorios. Esto se debe a que no se encontraron fallos (failures en inglés) ni errores (errors en inglés) por lo que la barra de estado es de color verde, de haber ocurrir algún error o algún fallo la barra cambiaría a color rojo.

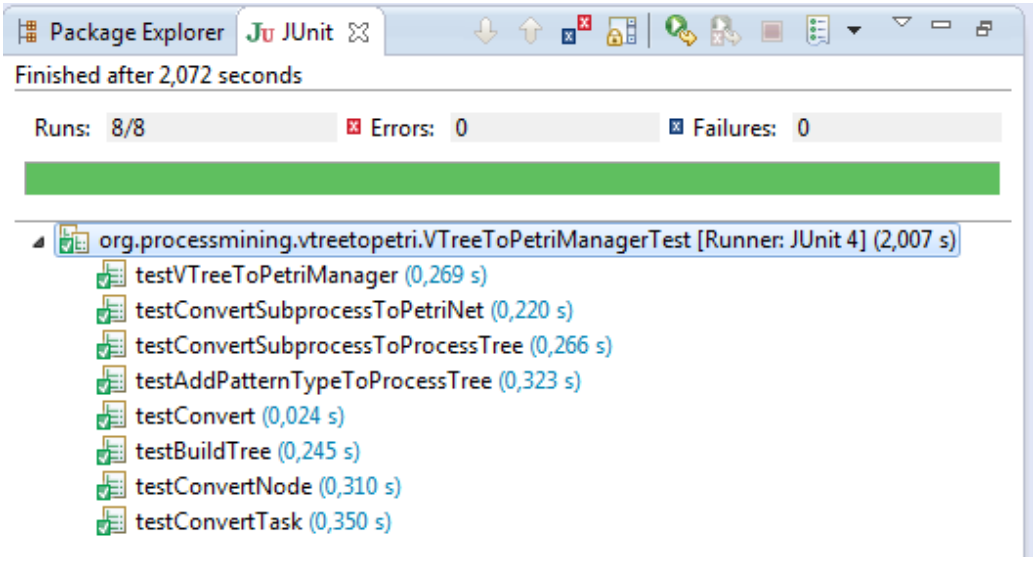


Figura 27: Interfaz de resultados de Junit.

3.2.2. Validación del funcionamiento del complemento "Convert Variant tree to Petri Net".

El funcionamiento del complemento se comprueba con cuatro Árboles de Variantes generados en el grupo de investigación, a partir de cuatro registros de eventos previamente seleccionados. Se generan las redes de Petri y se comprueba si el modelo obtenido, se corresponde con el modelo de Árbol de Variantes. La correspondencia entre ambos modelos se establece en términos de las actividades y los patrones de control de flujo presentes en los modelos. Para más información consultar el documento"*Casos de prueba del funcionamiento del complemento Convert Vtree To Petri Net*" dentro del expediente de investigación.

A continuación se muestran los patrones de control de flujo por los cuales se descomponen los subprocesos del Árbol de Variantes y su representación en notación red de Petri (Alfonso, y otros, 2014), y el primer caso de prueba realizado al complemento:

- **Secuencia (SEQUENCE):** Dos subprocesos se ejecutan secuencialmente (Figura 28) si uno ocurre inmediatamente después del otro.

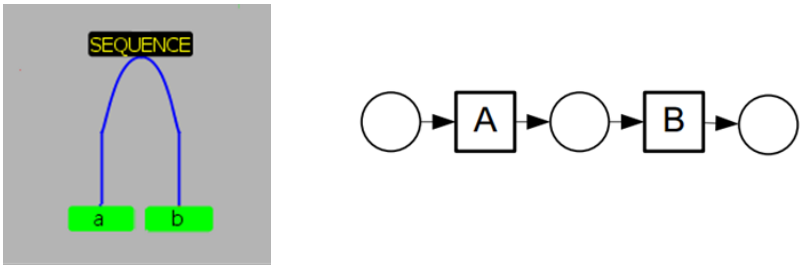


Figura 28: Secuencia en red de Petri.

- **Selección exclusiva (XOR):** dos subprocesos forman parte de una selección exclusiva (Figura 29) si, en un punto de decisión, se puede ejecutar solamente uno de los dos.

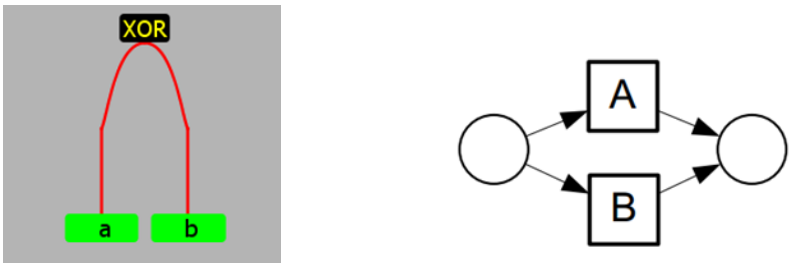


Figura 29: Selección exclusiva.

- **Paralelismo (PARALLELISM):** Dos subprocesos se ejecutan en paralelo (Figura 30) si ambos se ejecutan simultáneamente.

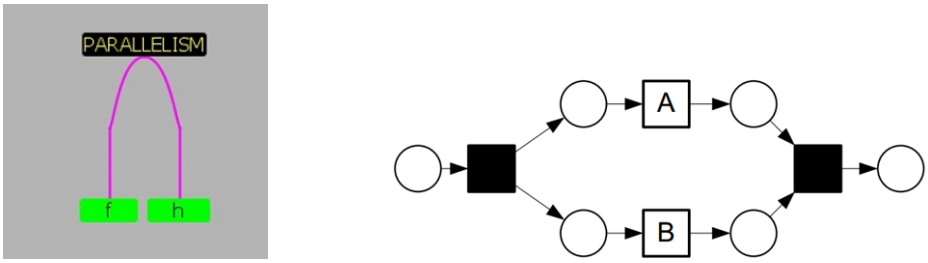


Figura 30: Paralelismo.

- **Lazo (LOOP):** Dos subprocesos se encuentran en un lazo (Figura 31) si se repiten múltiples veces. Cada repetición comienza con la ejecución del primer subproceso (Do), continúa con el segundo (Redo) y termina con el Do. El Redo puede ser un subproceso vacío, por lo que el único repetido sería el Do.

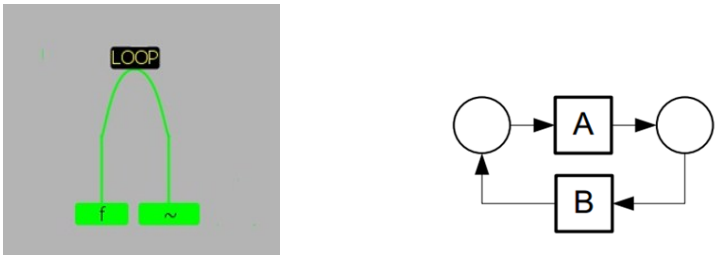


Figura 31: Lazo.

- **Selección no exclusiva (OR):** Dos subprocesos son opciones de una selección no exclusiva (Figura 32) si, en un punto de decisión, pueden ejecutarse ambos, o solamente uno de ellos.

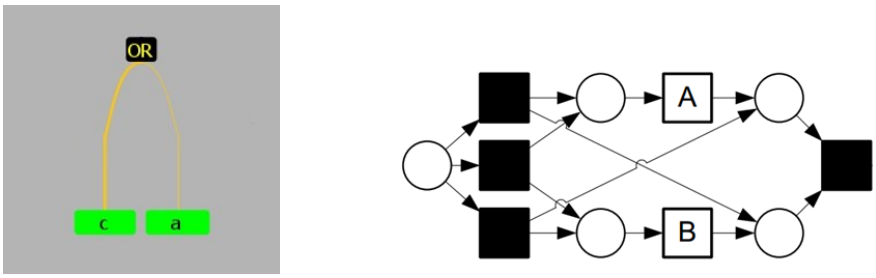


Figura 32: Selección no exclusiva.

Primer caso de prueba:

Al aplicar la técnica minería de variantes, al registro de eventos #1 se obtiene un Árbol de Variantes con tres niveles donde el nodo raíz es un subproceso de tipo *Secuencia*. Este tiene como nodos hijos en el segundo nivel del árbol un subproceso, la actividad *c* y otro subproceso. Ambos subprocesos son del tipo *Selección exclusiva*. Las actividades *a* y *b* son los nodos hijos del primer subproceso y del segundo son hijas las actividades *d* y *e;* las cuatro en el tercer nivel del árbol. El Árbol de Variantes correspondiente al registro de eventos # 1 se muestra en la Figura 33.

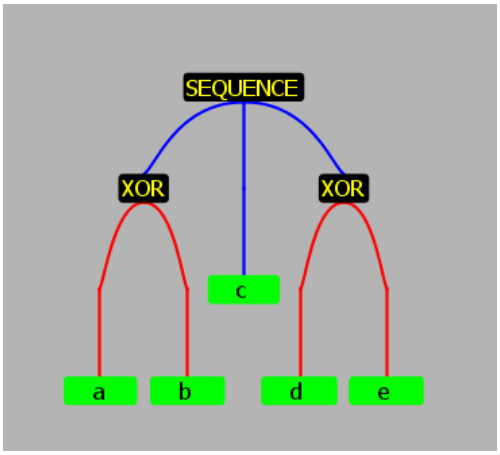


Figura 33: Árbol de Variantes del registro de eventos del primer caso de prueba.

Para la creación de la red de Petri se recorre el Árbol de Variantes desde el primer nivel hasta el último y de izquierda a derecha hasta recorrer todos los nodos del árbol. Los elementos anteriormente descritos pueden ser analizados a través de la notación: **Seq(Xor(a , b) , c , Xor(d , e))** que se muestra en la consola del eclipse en tiempo de ejecución**.** Esta notación permite verificar la coincidencia de los patrones y las actividades generados en la red de Petri con los del Árbol de Variantes**.** En la Figura 34 se representa la red de Petri generada por el complemento en correspondencia con el Árbol de Variante y la notación anteriormente mostrada. La red de Petri obtenida posee los mismos patrones de control de flujo y las mismas actividades. De esta forma queda demostrada la correspondencia entre ambos modelos.

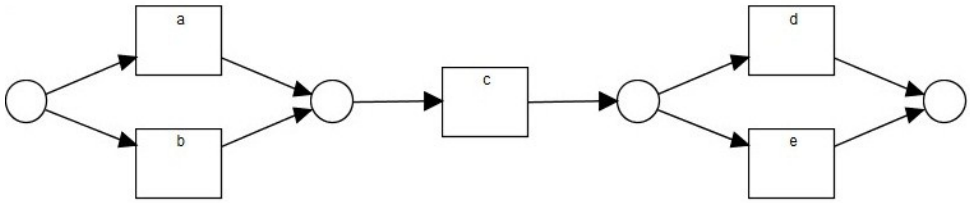


Figura 34: Red de Petri correspondiente al modelo de Árbol de Variantes del primer caso de prueba.

Segunda iteración:

EL primer Árbol de Variantes fue correctamente transformado a una red de Petri en la primera iteración, pero se aplicó una segunda iteración para comprobar que los cambios

realizados en la aplicación no afectaran el resultado, obteniéndose los mismos descritos en la primera iteración.

Resultados Generales:

En total se realizaron 4 pruebas junto con dos iteraciones en cada una. En la primera iteración se detectaron 5 errores entre la generación incorrecta de patrones y actividades. Para la segunda iteración los errores ya estaban corregidos. En la Figura 35 se muestra el resultado de los casos de pruebas en la primera iteración. El resultado de todas las pruebas se resume en la Figura 36, donde se puede apreciar la concordancia entre los tipos de patrones que existen entre los Árboles de Variantes y las redes de Petri de cada registro de eventos.

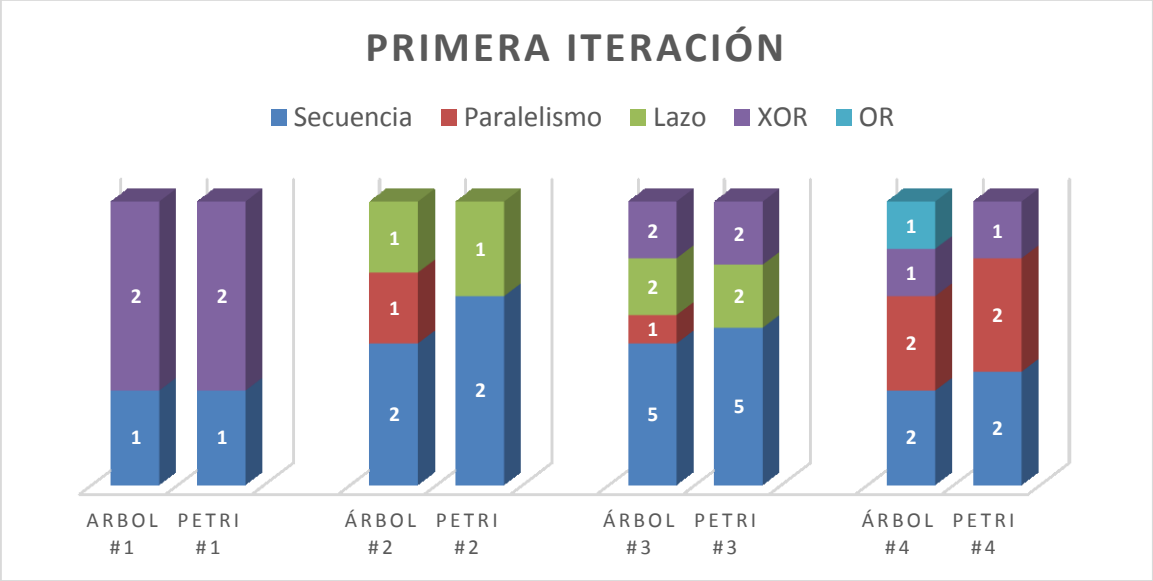


Figura 35 Coincidencia de patrones de control de flujo por cada árbol y su respectiva red de Petri equivalente en la primera iteración

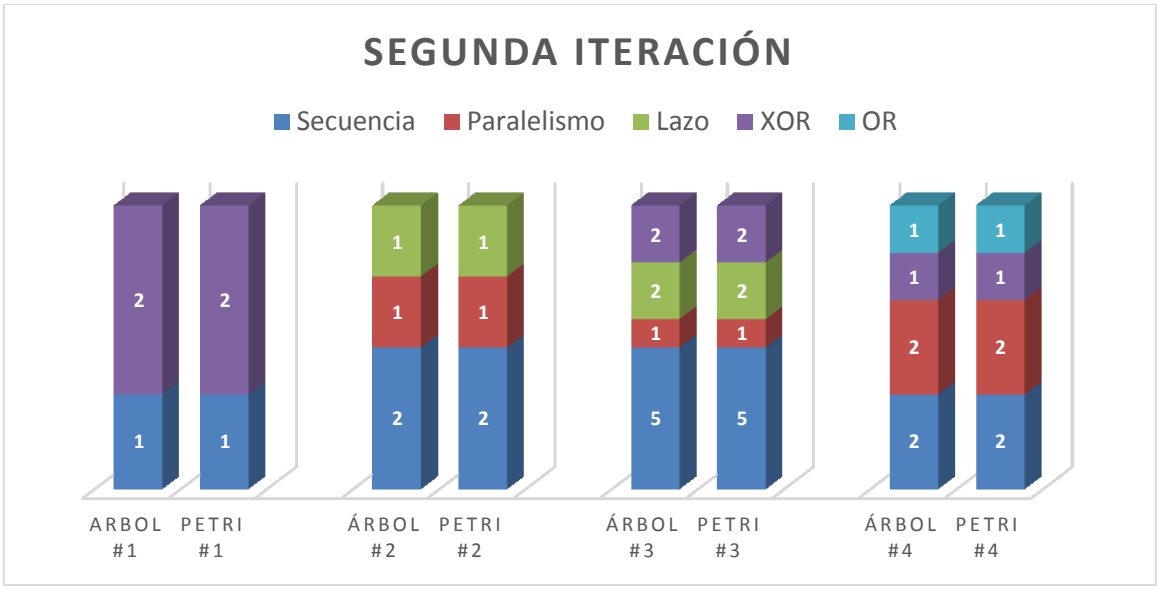


Figura 36: Coincidencia de patrones de control de flujo por cada árbol y su respectiva red de Petri equivalente en la segunda iteración

En la Figura 37 se muestra el resultado final de la cantidad de actividades mostrada por cada modelo con respecto a los registro de eventos de cada prueba.

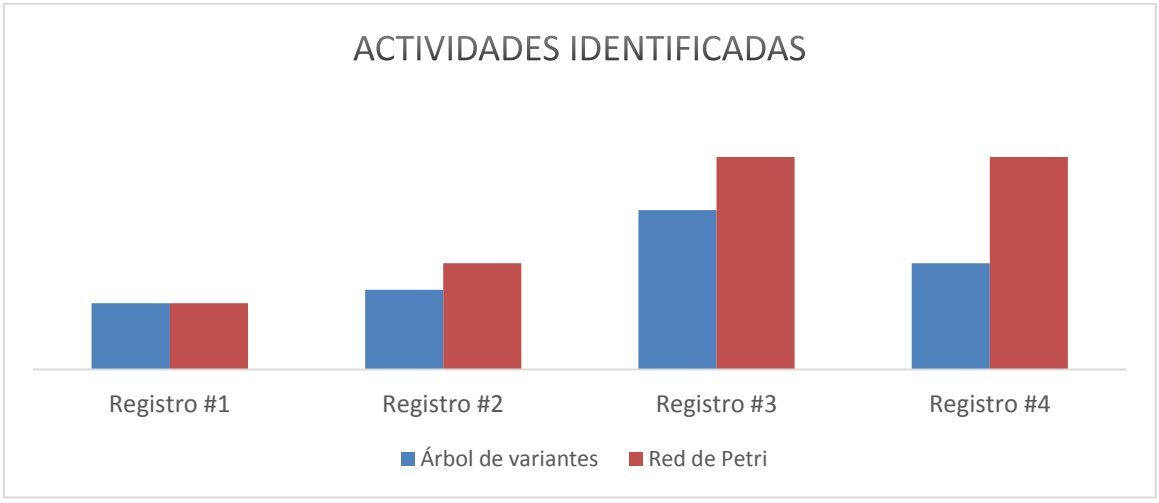


Figura 37: Cantidad de actividades en los Árboles de Variantes y en las redes de Petri por cada registro de evento.

Las redes de Petri en algunos casos presentan un mayor número de actividades que los Árboles de Variantes. Esto se debe a que en las redes de Petri es necesario insertar actividades invisibles para conectar algunos patrones de control de flujo, pues sin ellas el modelo obtenido no es correcto. Por tanto, la cantidad de actividades mostradas en una

red de Petri equivalente a un Árbol de Variantes tiene que ser mayor o igual a la cantidad mostrada por este último. Si en algún caso es menor, significa que en el modelo se omitió una actividad durante la transformación, por tanto deja de ser equivalente al Árbol de Variantes.

A partir de las pruebas realizadas se evidencia que las actividades y los patrones presentes en los Árboles de Variantes se identifican correctamente en las redes de Petri generadas con el complemento. Tomando en cuenta lo anteriormente planteado se llega a la conclusión de que la aplicación funciona correctamente

3.2.3. Aplicando métricas de calidad a las redes de Petri.

En la Minería de Procesos los modelos obtenidos se pueden evaluar de dos formas, modelo-modelo y modelo-registro de eventos. En la forma modelo-modelo se mide la coincidencia entre el modelo descubierto y el modelo de referencia. En la forma modelo-registro de eventos, se comprueba el modelo obtenido con el registro de eventos del cual fue descubierto (Ailenei, 2011).

Además, existen cuatro dimensiones para medir la calidad de los modelos de procesos descubiertos (van der Aalst, 2011) (Ma, 2012) (Broucke, y otros, 2013).

- **Aptitud:** Las métricas de esta dimensión evalúan la capacidad del modelo de representar el comportamiento presente en el registro de eventos (Broucke, y otros, 2013). En la Figura 38 se muestra el resultado de aplicar la métrica Fitness, donde el valor oscila entre 0 y 1, siendo 1 el valor deseado.

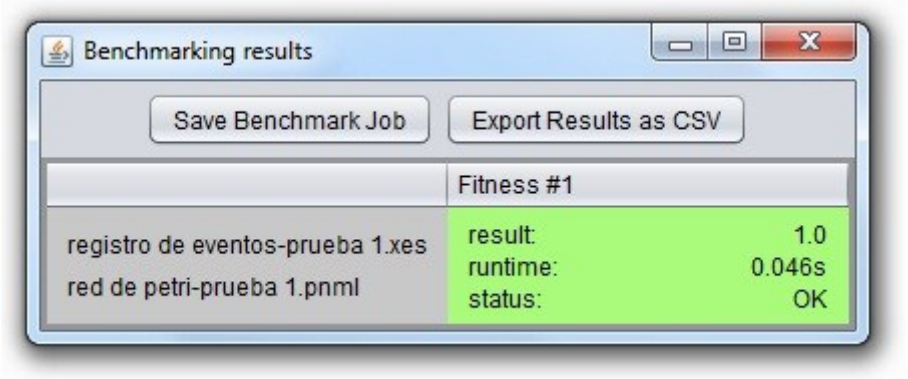


Figura 38: Resultado de la métrica Fitness

- **Precisión**: Las métricas de esta dimensión cuantifican el ajuste del modelo al comportamiento representado en el registro de eventos y su capacidad de no permitir ningún comportamiento que no esté registrado (van der Aalst, 2011) (Ma, 2012). En la Figura 39 se muestra el resultado de aplicar la métrica Precisión ETC (Técnica de evaluación de Conformidad, del inglés Evaluation Tecnique Conformance). El valor del resultado oscila entre 0 y 1, donde se espera que sea lo más cercano posible a 1.

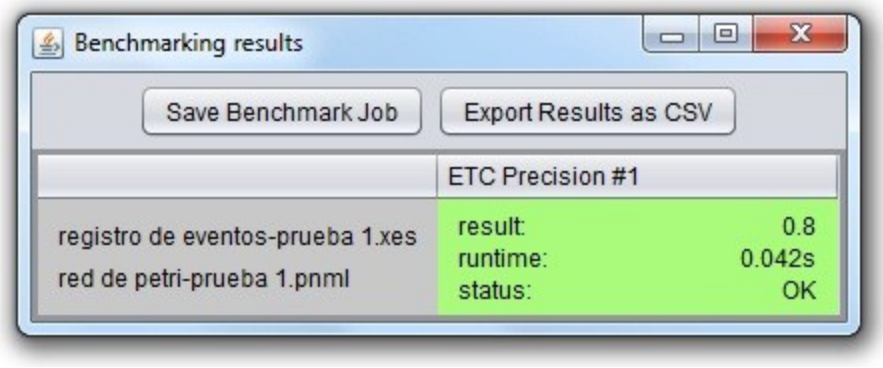


Figura 39: Resultado de la métrica Precisión ETC

- **Generalización:** Las métricas de esta dimensión miden las posibilidades del modelo de representar comportamiento no observado en el registro de eventos. Si el modelo está excesivamente ajustado al comportamiento observado, puede ser que otra muestra del mismo proceso genere un modelo completamente diferente. Por tanto, las métricas que cuantifican esta dimensión penalizan los modelos de procesos demasiados precisos (Broucke, y otros, 2013). En la Figura 40 se muestra el resultado de aplicar la métrica Behavioral Generalization (Generalización del Comportamiento). El valor del resultado oscila entre 0 y 1, donde 1 es el valor deseado.

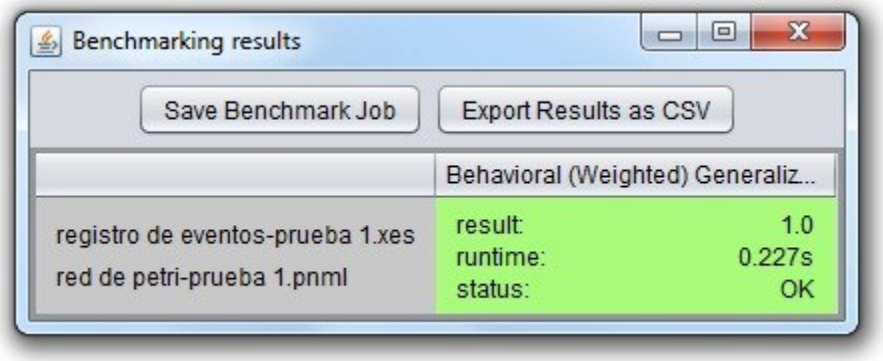


Figura 40: Resultados de la métrica Behavioral Generalization

- **Simplicidad:** Las métricas de esta dimensión miden que el modelo sea tan simple como sea posible, siempre que represente el comportamiento que está en el registro de eventos. La simplicidad de un modelo se expresa en el número de nodos y arcos que posee. Mientras más simple sea el modelo, este mejor se entiende. A medida que el modelo tenga más tareas duplicadas e invisibles, más penaliza las métricas en su evaluación. (Broucke, y otros, 2013). En la Figura 41 se muestra el resultado de aplicar la métrica Average Node Arc Degree (Media de nodo de grado de arco) donde el resultado el resultado esperado tiene que ser lo más cercano posible a 0.

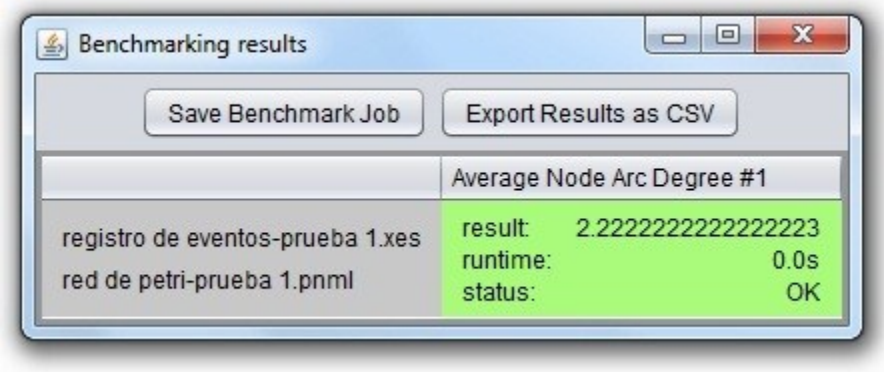


Figura 41: Resultado de la métrica Average Node Arc Degree

Se aplicó al menos una métrica por cada dimensión de calidad, esto prueba que los modelos obtenidos mediante el complemento son correctos y que es posible aplicar métricas de calidad sobre ellos.

3.2.4. Validación de la solución.

Para la validación de la solución se hizo una consulta a cuatro especialistas en el tema. Estos, a través de un cuestionario compartieron sus criterios y evaluaciones sobre los principales elementos de la investigación. A continuación se presenta en la Tabla 4 la categoría, nivel profesional y los años de experiencia de estos especialistas.

Especialistas	**Categoría docente**	**Categoría científica**	**Nivel profesional**	**Años de experiencias**
Especialista 1	Asistente	Doctor	Doctor	7
Especialista 2	Instructor	Sin categoría	Ingeniero	6
Especialista 3	Asistente	Sin categoría	Ingeniero	5
Especialista 4	Sin categoría	Sin categoría	Ingeniero	1

Tabla 4: Relación de especialistas

La aplicación del cuestionario a los especialistas arrojo los siguientes resultados:

- El sistema transforma un Árbol de Variantes en una red de Petri.
- El sistema permite reubicar los nodos en el modelo
- Se puede aplicar zoom al modelo
- Se puede exportar la red de Petri visualizada
- El modelo obtenido puede ser analizado por otros complementos

3.3. Conclusiones parciales.

En este capítulo para inspeccionar el código del software en su conjunto se realizaron pruebas dinámicas de caja blanca. Para ello se empleó el marco de trabajo Junit valiéndose de las funcionalidades, librerías e integración con el Eclipse, donde se mostraron resultados satisfactorios, de cero errores en el código.

Se examinaron las funcionalidades del software desarrollado mediante casos pruebas conformadas por varios Árboles de Variantes donde se corroboró la equivalencia que existe entre estos y las redes de Petri generada con el software desarrollado.

Se realizó una consulta a especialistas los cuales brindaron sus criterios a partir de los cuales se comprobó la validez y el cumplimiento de los objetivos de la investigación.

CONCLUSIONES GENERALES

El análisis teórico de la investigación posibilitó descubrir las deficiencias de la técnica Minería de Variantes, lo que demostró la necesidad de la implementación de un complemento que transforme un Árbol de Variantes en una red de Petri para mitigar estas deficiencias.

El empleo de la metodología AUP sirvió para guiar el proceso de desarrollo del software, donde se generaron los artefactos necesarios para la creación del mismo.

Las métricas de validación del diseño y patrones empleadas en el desarrollo del software permitieron llegar a resultados fiables del diseño de la aplicación.

La realización de pruebas de caja blanca y casos de pruebas funcionales posibilitó evaluar el software en su conjunto. Además, la aplicación de las métricas de calidad a las redes de Petri permitió verificar la calidad de las mismas.

Se desarrolló un complemento que transforma un Árbol de Variantes en un modelo equivalente en notación de red de Petri, permitiendo aplicar métricas de calidad de minería de proceso, resolviendo de esta forma el problema de la investigación y cumpliendo con los objetivos propuestos.

RECOMENDACIONES

Hacer un profundo estudio para determinar si es posible agregar nuevas funcionalidades que permitan generar en una misma interfaz las redes de Petri de todas las variantes encontradas con la técnica minería de variantes.

BIBLIOGRAFÍA

—. 1994. *Putting high-level Petri nets to work in industry.* Eindhoven University of Technology, Department of Mathematics and Computing Science : s.n., 1994.

Aalst, W.M.P. van der. 1995. *A class of Petri nets for modeling and analyzing business processes.* Eindhoven University of Technology, Department of Mathematics and Computing Science : s.n., 1995.

Aalst, Wil M.P. van der. 2013. *Process Cubes: Slicing, Dicing, Rolling Up and Drilling Down Event Data for Process Mining.* 2013.

Abran, Alain , et al. 2004. *SWEBOK: Guia al cuerpo de conocimiento de la ingenieria del software.* California : IEEE COMPUTR SOCIETY, 2004. ISBN 0-7695-2330-7.

Ailenei, Irina Maria. 2011. *Process Mining Tools: A Comparative Analysis.pdf.* EINDHOVEN UNIVERSITY OF TECHNOLOGY : s.n., 2011.

Alfonso, D. P., et al. 2014. *Búsqueda de costo uniforme para el diagnóstico de variantes de procesos de negocio.* La Habana, Cuba : I Conferencia Científica Internacional UCIENCIA 2014, 2014.

Ambler, Scott. 2014. Effective Practices for Software Solution Delivery. [Online] 2014. [Cited: 4 9, 2014.] http://www.ambysoft.com/.

Arnow, David and Weiss, Gerald. 2001. *Introducción a la Programación Java.* s.l. : Pearson Publications Company, 2001.

Broucke, vanden , et al. 2013. *A Comprehensive Benchmarking Framework (CoBra) for Conformance Analysis between Procedural Process Models and Event Logs in ProM.* Brisbane, Australia : Queensland University of Tecnology, 2013.

Chandra Bose, Rantham Prabhakara Jagadeesh and van der Aalst, Wil . 2010. *Trace Alignment in Process Mining: Opportunities for Process Diagnostics.* Eindhoven : s.n., 2010.

Eclipse. 2010. Proyecto Eclipse. [Online] julio 8, 2010. [Cited: 4 8, 2014.] http://www.eclipse.org.

Fakhroutdinov, Kirill. 2014. The Unified Modeling Language. [Online] Object Management Group, 5 26, 2014. [Cited: 5 26, 2014.] http://www.uml-diagrams.org/component-diagrams.html.

Gamma, Erich, et al. 2002. *Design Patterns: Abstraction and Reuse of Object-Oriented Design.* University of Illinois : Department of Computer Science, 2002. ISBN 0-201-63361-2.

Günther, Christian W. 2009. *Process Mining in Flexible Environments.* Eindhoven : Eindhoven University of Technology Library, 2009. ISBN 978-90-386-1964-4.

Hommel, Scott. 1999. *Convenciones de Código para el lenguaje de programación JAVA.* s.l. : Sun Microsystems Inc., 1999.

Jacobson, Ivar, Booch, Grady and Rumbauch, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison Wesley, 2000. ISBN:84-7829-036-2.

Larman, Craig. 2005. *UML y Patrones: Una introducción al análisis, el diseño orientado a objetos y al proceso unificado. 2da edición.* s.l. : Prentice Hall, 2005.

Ma, Lulu. 2012. *How to Evaluate the Performance of Process Discovery Algorithms.* Eindhoven University of Technology : s.n., 2012.

Magliano, Virginia María, Bazán, Patricia and Martinez Garro, José. 2013. *Análisis metodológico para la utilización de Process Mining como tecnología de optimización y respaldo de la implementación de procesos de negocio bajo el marco de BPM.* Buenos Aires, Argentina : Facultad de Informática UNLP, LINTI Facultad de Informática UNLP, 2013.

Mamaliga, Tatiana. 2013. *Realizing a Process Cube Allowing for the Comparison of Event Data.* Eindhoven : s.n., 2013.

Murata, Tadao. 1989. *Petri nets:Porperties, Analisis and Aplications.* s.l. : IEEE, 1989.

Openbravo. 2007. OpenbravoERP. [Online] 5 4, 2007. [Cited: 6 14, 2014.] www.openbravo.com/es/.

Paradigm, Visual. 2012. Visual Paradigm. [Online] 4 6, 2012. [Cited: 3 12, 2014.] http://www.visual-paradigm.com/product/vpuml/editions/standard.jsp..

Patton, Ron. 2006. *Software Testing.* Indianapolis : SAMS, 2006. ISBN 0-672-31983-7.

Pressman, Roger S. 2005. *Ingenieria de Software: Un enfoque practico. Quinta Edicion.* s.l. : McGraw Hill, 2005.

Rainsberger, J. B. 2005. *Junit Recipes: Practicl Methods for programmer Testing.* s.l. : Manning Publications, 2005.

Roest , Harment. 22 November 2012. *A PRACTITIONER'S GUIDE FOR PROCESS MINING ON ERP SYSTEMS – THE CASE OF SAP ORDER TO CASH.* Eindhoven : s.n., 22 November 2012.

Weske, Mathias. 2007. *Business Process Management: Concept, Languages, Architectura.* Potsdam : s.n., 2007. ISBN 978-3-540-73521-2.

Shaikh, Yasmin . febrero,2013. *Internationa Journal of Engenieering Research and Science y Technology.* Hyderabad,India : IJERST, febrero,2013. ISBN 2319-5991.

Sommerville, Ian. 2005. *Ingeniería de Software.* Madrid : Addison Wesley, 2005. ISBN:84-7829-074-5.

Song, Minseok and van der Aalst, Will. 2007. *Supporting process mining by showing events at a glance.* Montreal : s.n., 2007.

Ultimus. 2012. La Suite de Software BPM Adaptativa de Ultimus. *La Suite de Software BPM Adaptativa de Ultimus.* [Online] 2012. [Cited: 6 14, 2014.] http://www.ultimus.com/es/.

van der Aalst, Wil M.P. and Chandra Bose, Jagadeesh . 2009. *Abstractions in Process Mining: A Taxonomy of Patterns.* Eindhoven : s.n., 2009.

van der Aalst, Wil, et al. 2011. *Manifiesto sobre Mineria de Procesos.* s.l. : IEEE Task Force on Process Mining, 2011.

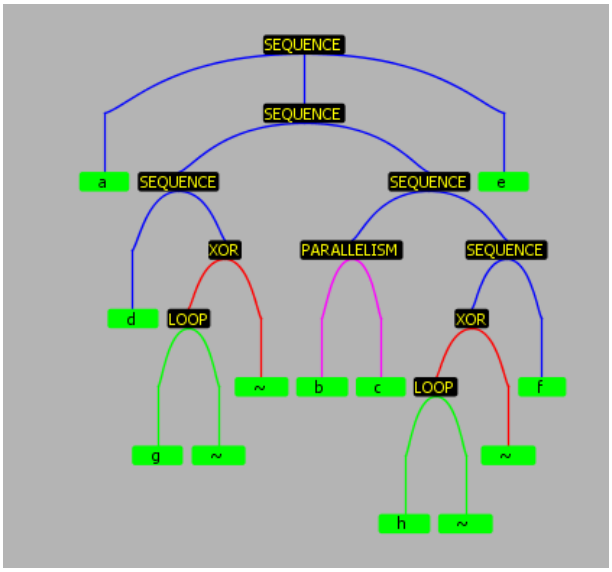
van der Aalst, Will M.P. 2011. *Process Mining:Discovery, Conformance and Enhancement of Business Processes.* Eindhoven : s.n., 2011. ISBN 978-3-642-19344-6.

van Doremalen, Bart . 2012. *Process Mining in Healthcare Systems: An Evaluatio and Refinement of a Methodology.* Eindhoven : s.n., 2012.

Verbeek, Eric and W. Günther, Christian . 2010. ProM. *Process mining workbench.* [Online] Process Mining Group,Eindhoven Technical University, 2010. [Cited: 4 7, 2014.] http://www.promtools.org/prom6/prom62.html.

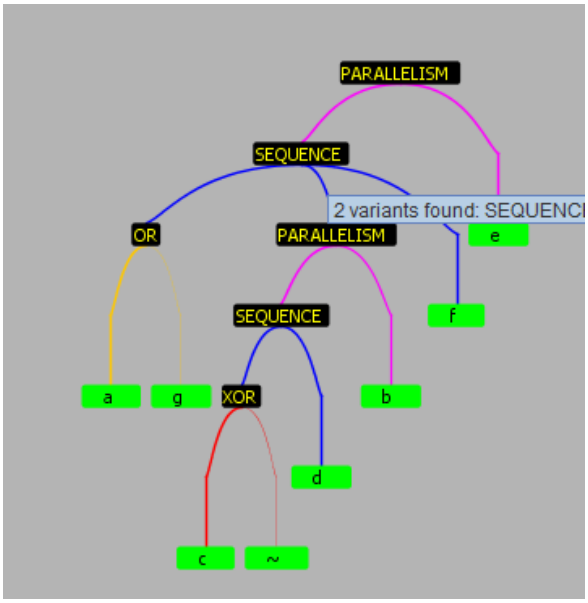
ANEXOS

Anexo 1: Árbol de Variantes prueba 3.



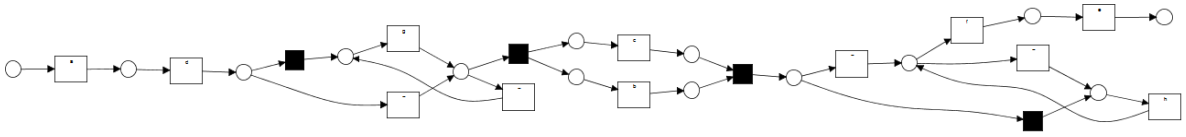
Anexo 1: Árbol de Variantes prueba 3

Anexo 2: Árbol de Variantes prueba 4.



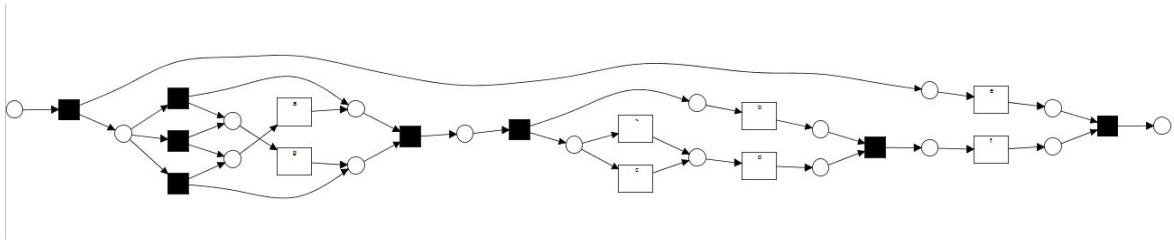
Anexo 2: Árbol de Variantes prueba 4

Anexo 3: Red de Petri prueba 3.



Anexo 3: Red de Petri prueba 3

Anexo 4: Red de Petri prueba 4.



Anexo 4: Red de Petri prueba 4

Anexo 5: Preguntas del cuestionario.

Preguntas:

1. **¿El sistema transforma un árbol de variantes en una red de Petri?**
2. **¿Se pueden reubicar los nodos en el modelo?**
3. **¿Se puede aplicar zoom al modelo?**
4. **¿Se puede exportar la red de Petri visualizada?**
5. **¿El modelo obtenido puede ser analizado con otros complementos?**

Anexo 5: Preguntas del cuestionario