UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

TRANSFORMACIÓN DE DIAGRAMAS DE COMPORTAMIENTO A MODELOS DE CONCURRENCIA

Antonio Oliva Hernández
Tutor: María Elena Gómez-Martínez
Ponente: Juan de Lara Jaramillo

JULIO 2020

TRANSFORMACIÓN DE DIAGRAMAS DE COMPORTAMIENTO A MODELOS DE CONCURRENCIA

AUTOR: Antonio Oliva Hernández
TUTOR: María Elena Gómez-Martínez

Grupo de Modelado e Ingeniería del Software
Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2020

Resumen

Este Trabajo Fin de Grado explora el uso de transformaciones de modelos de máquinas de estados a redes de Petri estocásticas generalizadas en el entorno de desarrollo de ingeniería de rendimiento software, y detalla la creación de una implementación del estudio teórico en una plataforma de desarrollo de software popular a través de ingeniería orientada a modelos.

El estudio teórico se centrará en el tipo de modelos de concurrencia conocido como redes de Petri. Este modelo permite describir los modelos de comportamiento de sistemas comunes mediante diagramas abiertos a extracción de métricas y análisis, ampliando los horizontes de diseño de autómatas de estados finitos, una de las representaciones de comportamiento de sistemas más comúnmente utilizadas.

Dichas transformaciones pues, conservarán todas propiedades del diagrama de modelado de la máquina de estados, añadiendo las ventajas del nuevo formalismo, así como útiles adicionales como medidas temporales y de control de recursos.

La implementación buscará la maximización de la accesibilidad y agilidad de descripción de los sistemas diseñados por el usuario y facilitándole acceso a plataformas de análisis más completos sin requerir complejos diseños iniciales.

El desarrollo de la aplicación estará basado en herramientas de desarrollo de software establecidas como el sistema de transformaciones de modelos ATL, desarrollada por la compañía vendedora de herramientas de software francesa OBEO y el Instituto Nacional francés de Investigación en Informática y Automática; el generador de código y plantillas Acceleo, desarrollado por la Fundación Eclipse; así como herramientas para el manejo de redes estocásticas de análisis como GreatSPN, desarrollada por la Universidad de Turín.

Con estas herramientas estandarizadas, la aplicación marcará como objetivo ser lo más modular posible y compatible con otras, permitiendo expansiones del proyecto o acoplamiento a otros proyectos de mayor escala.

Palabras clave

Máquina de estados, redes de Petri estocásticas, ingeniería de prestaciones software, ingeniería orientada a modelos, transformaciones de modelos, UML, MARTE.

Abstract

This Bachelor Thesis explores the use of model transformations from state machine diagrams to generalized stochastic Petri nets through a software performance engineering approach. It also details the creation of an implementation of the researched topics on a common software development platform through model driven engineering.

The theoretical part of the paper will focus on the concurrency models known as Petri nets, developed in the first half of the last century. This model allows for definitions of common behavioural system models through diagrams open to the extraction of performance metrics and further analysis. Such definitions allow for richer designs for finite state automata, one of the most popular behavioural system representations in engineering.

Such transformations will aim to conserve all the properties of the original finite state machine model diagram, adding new details inherent to the new host model, as well as extra metrics such as time and resource management.

The implementation will seek to maximize accessibility and agility when defining user designed system models and grant the user access to more complete analysis tools without requiring a high level of initial technical knowledge from the user.

The development of the application will be based on established software development tools such as the model transformation system ATL, developed by the French software vendor company OBEO, and the French National Institute for Research in Computer Science and Automation; the template and code generator tool Acceleo, developed by the Eclipse Foundation; as well as stochastic Petri net management tools such as GreatSPN, developed by the university of Turin.

Thanks to these standardized tools, the developed application will set its goal in becoming as modular and compatible with other applications as possible, allowing for further expansions or a modular fit into other projects of greater scale.

Keywords

State machine, stochastic Petri net, software performance engineering, model driven engineering, model transformations, UML, MARTE.

Agradecimientos

Quisiera agradecer el apoyo, paciencia y confianza de mi tutora, Elena Gómez Martínez, y también por darme la oportunidad de trabajar en este proyecto.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Los diagramas UML[21] son una excelente tecnología utilizada frecuentemente en ingeniería del software, no obstante, a menudo no es suficiente para describir todas las características de un sistema o sus métricas de rendimiento. Esta memoria de TFG expondrá una solución al problema mediante transformación de diagramas UML a otros modelos más favorables a ejecución de análisis y una implementación en Eclipse[9] de la misma.

Específicamente, se explorará el estudio de máquinas de estado para el estudio de comportamiento de sistemas, y su transformación a redes de Petri para evaluación de recursos y concurrencia en tiempo de diseño.

Para fabricar esa representación en una aplicación práctica, existe un paradigma de transformaciones a través de herramientas estandarizadas, la ingeniería orientada a modelos, que se empleará en el diseño de la transformación de este trabajo.

Asimismo, por maximizar la utilidad del software desarrollado, así como facilidad para expandir el proyecto, se hará uso de la mayor cantidad de herramientas ya desarrolladas y establecidas. Eclipse cuenta con numerosas herramientas de desarrollo de software dentro del entorno y es uno de los entornos de desarrollos más populares.

En este trabajo de investigación se detalla la serie de herramientas de transformación a modelos de redes de Petri estocásticas generalizadas para obtener métricas de rendimiento integradas en Eclipse desarrolladas, tomando los principios de la ingeniería de rendimiento del software (*Software Performance Engineering*, SPE)[11].

Finalmente, se elaborará un conjunto de pruebas y guías de validación de la transformación, así como la extracción de dichas métricas, detalladas en la sección de resultados de esta memoria.

1.2 Objetivos

Los principales objetivos que se propone el proyecto vienen descritos por la raíz del problema. Estos son:

- El uso de herramientas estándar de procesamiento de modelos para la generalización de modelos de máquinas de estado y su traducción a modelos Ecore de Eclipse EMF.

- La generación de modelos de redes de Petri conformes a modelos de máquinas de estados conservando la estructura funcional de la máquina.

- El estudio de simulaciones y documentación de métricas de ejecución de dichos modelos.

1.3 Organización de la memoria

Este trabajo de divide en siete capítulos, el apartado de referencias, el glosario y cuenta además con 3 anexos. A continuación, se describirá el contenido de cada uno de ellos.

En el primer capítulo se presenta una visión general de la investigación, se plantea el problema a investigar y los objetivos del trabajo.

En el segundo capítulo se exponen los conocimientos teóricos básicos necesarios para entender el contexto del problema.

En el tercer capítulo se aborda el estado de la cuestión. Se comienza estudiando el entorno actual de la programación de modelos, el estudio de las transformaciones entre modelos, las medidas de rendimiento de los sistemas de máquinas de estados, así como las herramientas actuales y entornos de programación para el desarrollo de dichas transformaciones.

En el cuarto capítulo se detallan las herramientas seleccionadas para el trabajo, así como los conceptos y tecnologías abordadas.

En el quinto capítulo se realiza un despliegue detallado de los pasos para cada transformación, la organización de archivos utilizados, los elementos transformados, así como el método de generación de los archivos de salida y las métricas evaluadas.

En el sexto capítulo se describen los métodos de validación de la implementación, evaluación del progreso y resultado, y se enumeran los ejemplos sobre modelos de máquinas de estados reales producidos y sus métricas generadas.

Finalmente, en el séptimo y último apartado se declara la evaluación del proceso realizado, su utilidad y contribución al entorno actual de la programación mediante modelos, así como la exploración de posibles puntos de expansión para nuevos horizontes de ampliación de prestaciones del proyecto.

El primer anexo consiste en la descripción de las herramientas e instrucciones necesarias para la correcta instalación para utilizar el software como usuario. Esto incluye los paquetes de Eclipse, librerías y ficheros adicionales.

El segundo anexo consta de las instrucciones y herramientas adicionales para acceder al software como desarrollador, para expandir sus capacidades o adaptarlas a otro entorno o funcionalidad.

2 Marco teórico

A continuación, se expone una serie de conceptos teóricos básicos sobre los lenguajes de modelado y prácticas de diseño que no dependen de la implementación, pero sirven de soporte a los siguientes capítulos.

2.1 Máquinas de estados

Las máquinas de estados finitos son un modelo matemático[30] de descripción de comportamiento de sistemas en el cual el sistema, descrito como una máquina abstracta puede encontrarse en una de varias posibles configuraciones, o estados, y los pasos de un estado a otro describen el comportamiento del sistema. Las máquinas de estado son comúnmente utilizadas por su utilidad a la hora de describir la gran cantidad de simples autómatas que construimos y utilizamos cada día.

2.2 Introducción a redes de Petri

Las redes de Petri[23] son un lenguaje matemático de modelado de sistemas distribuidos. El lenguaje está basado en un esquema de grafo bipartido, con los dos conjuntos de vértices llamados lugares y transiciones respectivamente, y las aristas siendo denominadas arcos.

Una red de Petri representa sistemas mediante el paso de objetos a través de los elementos gramaticales de la red, con cada iteración del paso representando una instancia inmediata o un paso temporizado. La cantidad y tipo de objetos necesaria para avanzar un paso en la red, así como la frecuencia de dichos avances quedan definidos por las propiedades de los elementos de la red.

2.2.1 Lugares

Un lugar representa una posición estática en el sistema en el sistema. Cada lugar puede englobar una propia cantidad de objetos que han alcanzado el lugar. Los lugares son representados por círculos.

2.2.2 Transiciones

Una transición representa un paso en la ejecución de la red de Petri. Una transición puede recibir una serie de arcos de entrada, y una serie de arcos de salida. Una transición se representa con una barra o rectángulo. Cuando una transición recibe la serie de objetos definida desde cada arco de entrada, la transición se ejecuta, en un proceso llamado regla de disparo, consumiendo esa cantidad de objetos, y produciendo una serie de objetos de salida definida a través de cada arco de salida.

En el modelo más básico de redes de Petri, las transiciones son inmediatas. Existen otros modelos expandidos de redes de Petri que añaden nuevas características de la red y sus elementos. El más comúnmente utilizado es el modelo de redes de Petri estocásticas generalizadas (GSPN[21]).

En el modelo GSPN, existen varios tipos de transiciones, dependiendo de su capacidad de procesamiento de entradas y producción de salida. Entre ellas y relevantes

durante el desarrollo de este proyecto, se han incluido las transiciones inmediatas y de tiempo constante.

Una transición inmediata produce los objetos de salida en cuanto recibe los objetos por parte de los lugares procedentes de los arcos de entrada. En el caso de que un lugar tenga objetos que cumplan múltiples transiciones inmediatas salientes, la probabilidad de consumir cada objeto en cada transición particular viene definida por la prioridad de cada transición.

En el modelo GSPN, una transición temporizada tarda una cierta cantidad de tiempo en procesar los objetos provenientes de los arcos de entrada y producir los objetos para los arcos de salida. Durante el procesamiento temporizado, los objetos provenientes de los arcos de entrada no están ya disponibles en sus lugares de origen para consumo en otras transiciones.

2.2.3 Arcos

Un arco representa una unión direccionada entre un lugar y una transición, o viceversa. El número de objetos que deben atravesar un arco viene dado por el peso del arco.

2.2.4 Marcado inicial

A la hora de realizar una simulación, se pueden explorar distintas configuraciones iniciales de la red, produciendo distintos resultados finales. Dichas configuraciones iniciales pueden variar en parte por la cantidad de objetos iniciales a simular dentro de la red, así como su posición en la red.

2.2.5 Reglas de disparo

El conjunto de condiciones que definen un paso de una simulación de red de Petri se llama regla de disparo.

Las reglas de disparo incluyen, entre otras medidas, el número de objetos necesarios de entrada por cada arco, la duración de la transición, en particular para decidir si la transición ha finalizado su anterior transición, así como las probabilidades de disparo de transiciones inmediatas.

2.2.6 Bucles de desvanecimiento

Adicionalmente, existe una situación que impide simular el funcionamiento de una red de Petri, que se debe tener en cuenta al añadir la funcionalidad temporal al modelo para evitar crear diseños incompletos. Esta situación viene descrita por los siguientes sucesos:

Un lugar o lugares con marcado inicial superior a 1, pero conectado/s a una transición inmediata, cuya regla de disparo se puede cumplir en el estado inicial, el estado inicial de la red de Petri designado es equivalente a un estado con esa transición ejecutada. Este tipo de marcado inicial se llama marcado de desvanecimiento, porque nunca es analizado, o nunca existe en términos de la simulación, sino que se analiza la red una vez resueltos todos los marcados de desvanecimiento.

Esto no supone un problema a la hora de simular la red al final una vez resueltos los marcados de desvanecimiento, pero obviamente existe un caso en el que no es posible resolverlos, en el cual haya una cadena de marcados de desvanecimiento cerrada y no

disminuya el número de objetos, es decir, pueda repetirse infinitamente con como mínimo un número de objetos constante. Este suceso se llama un bucle de desvanecimiento. En la Figura 2-1 se puede mostrar un ejemplo de un bucle tal.

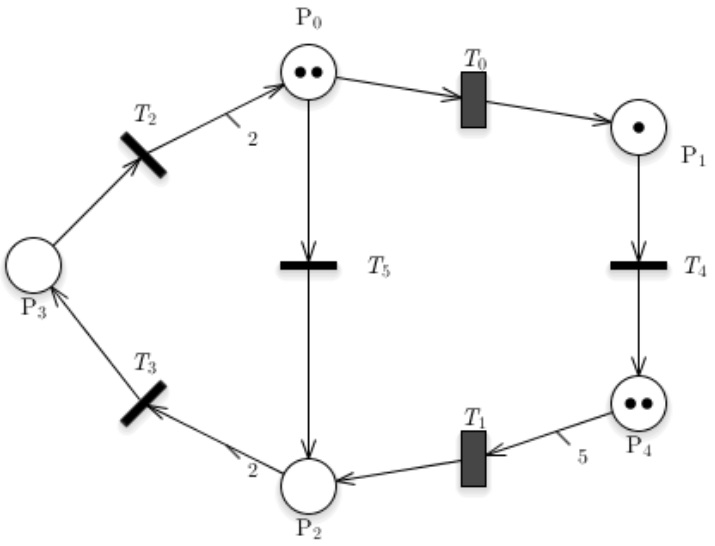

Figura 2-1: Bucle de desvanecimiento

Como se puede observar, el lado derecho de la red de Petri consume todos los objetos y llegaría a un estado final donde no se pueden cumplir más reglas de disparo. Esto es debido a que la transición T_1 requiere 5 objetos del lugar P_4, que se cumpliría si todos los objetos tanto de P_0 como de P_1 realizan transiciones por T_0 y T_4 para llegar a P_4. Una vez llegados los 5 objetos a P_4, podría dispararse la transición T_1, consumiendo los 5 objetos y produciendo un único objeto de salida en P_2. Una vez en P_2, no se puede disparar otra transición porque la única transición de salida de P_2, T_3, requiere 2 objetos, y sólo se tiene acceso a uno, llegando a un estado final.

No obstante, se puede observar que el marcado inicial del lugar P_0 es un marcado de desvanecimiento, pues puede ejecutarse la transición T_5 antes de ejecutar la simulación sin alterar la simulación, pues sería el primer paso de la simulación. Similarmente, el siguiente paso sería la inmediata transición de los dos objetos que han llegado a P_2 a través de T_3 a un único objeto en P_3, por lo que un marcado superior a 1 en P_2 sería un marcado de desvanecimiento, siendo este el caso. Por último, con el mismo razonamiento se puede declarar P_3 como otro lugar con marcado de desvanecimiento, volviendo al estado inicial descrito en la Figura 2-1, creando un bucle de desvanecimiento en el que no hay un estado inicial único de la simulación, y por tanto no puede comenzar una simulación.

Si por ejemplo fuera un arco de entrada de peso 3 de P_2 a T_3, de peso 1 el arco de T_2 a P_0 o 0 o 1 el marcado inicial de P_0, no habría bucle de desvanecimiento y se podría iniciar una simulación.

Evitar este tipo de bucles es una parte importante de diseñar un sistema con redes de Petri temporizadas.

2.3 Introducción a la ingeniería orientada a modelos

La ingeniería orientada a modelos (MDE), es un paradigma de desarrollo de software, basado en la creación y operación sobre modelos estandarizados, agilizando y reduciendo la complejidad del desarrollo, y permitiendo una mayor compatibilidad con otros sistemas, debido a la estandarización de los componentes.

El uso de este paradigma está basado en la existencia de una arquitectura inferior diseñada para incorporar este tipo de desarrollo simplificado, permitiendo al usuario familiarizado con el sistema a desarrollar modelarlo dentro de la arquitectura base. Este proceso se puede instanciar las veces que sean convenientes, creando una cadena de arquitecturas o metamodelos de dominio y complejidad cada vez menor para diseñar las funcionalidades específicas necesarias. En el contexto de este proyecto, se usará el MDE EMF de Eclipse, para crear un sistema MDE basado en el modelo de dominio simplificado UML.

3 Estado del arte

3.1 Estudios teóricos de la transformación

Existen una gran cantidad de estudios teóricos en el campo de transformaciones de modelos de máquinas de estados a redes estocásticas, aunque dependiendo de las necesidades del análisis se realizan también transformaciones de UML a otros tipos de modelos formales, como redes de colas por Cortellessa y Mirandola [28] o álgebras de procesos por Remenska, et al. [29]. Debido a las limitaciones inherentes del metamodelo UML, a menudo se fuerza el uso de herramientas externas para expresar propiedades como condiciones temporales, el uso de recursos o dependencias adicionales entre otras. Por ello es común el análisis de modelos expresados como máquinas de estados a través de una conversión a redes de Petri, un metamodelo más apto para el análisis de dichos recursos requeridos del sistema.

Se han desarrollados estos estudios para el análisis de proyectos de gran escala, como por ejemplo el estudio del sistema de control ferroviario europeo para coordinación de trenes y las estructuras de control de vías y diseño de las mismas por parte de Trowitzsch y Zimmerman [1].

También cabe destacar el estudio de Simon y Stoffel [13] utilizando la transformación para facilitar el mantenimiento de sistemas simplificando el trabajo de descripción de estos y creando un método más profundo de análisis y validación de los componentes y funcionalidades.

Asimismo, hay varios estudios realizados expandiendo la transformación, exportando la salida como una red de Petri coloreada[2][3][4]. Este tipo de modelos expanden el control de recursos de la red, permitiendo el paso de información particular a cada objeto de la red, como si fuera un objeto de programación convencional.

3.2 Herramientas de transformación

3.2.1 Fuera de Eclipse

Existen varios estudios en el campo de transformaciones de modelos de máquinas de estados a redes estocásticas, pero la mayoría de ellos no están implementados.

Existe un estudio con una implementación en Java, realizado por Wuytjens [5]. El estudio detalla una transformación similar, de sistemas descritos mediante máquinas de estados a modelos de redes de Petri, junto a una pequeña implementación como aplicación independiente. Respecto a la aplicación desarrollada, el paradigma tiene una desventaja clara, el potencial para expandir la aplicación. Al utilizar una aplicación externa en vez de una integrada en entornos de desarrollo, con puntos de entrada y salida, se impide la utilización de la aplicación en proyectos de alcance mayor. Además, las aplicaciones desarrolladas para Eclipse son cómodas de instalar y configurar, facilitando su uso y cohesión con el resto de software siendo desarrollado con la menor pérdida de tiempo posible.

Cabe destacar la existencia de un exhaustivo estudio realizado por Gómez-Martínez y Merseguer [27] en un ámbito muy similar al de este proyecto. El estudio explora la transformación de diagramas UML producidas por la herramienta ArgoUML a redes de Petri GSPN por medio de una aplicación desarrollada en Java llamada ArgoSPE. La aplicación queda contenida como una herramienta más dentro del editor de ArgoUML, pero, a diferencia del proyecto descrito por este trabajo, no realiza la transformación a través de programación MDE.

3.2.2 Implementadas dentro de Eclipse

Eclipse permite desarrollar aplicaciones dentro de la plataforma, facilitando integración con el desarrollo de nuevas aplicaciones, actuando como un componente más.

Existe una cantidad considerablemente más pequeña de implementaciones de estas transformaciones en el entorno de programación de Eclipse, mayormente con la metodología MDE debido a las herramientas adicionales que facilita el entorno.

Entre ellas se puede encontrar una implementación exclusivamente con Acceleo por parte de André et al. [7]. No obstante, tal y como indica el documento y la razón por la que se ha incluido ATL en el proceso de transformación, Acceleo, como exportador de plantillas, es bastante limitado en modificación de elementos, y aunque cumple una función importante en el proceso, realizar la transformación exclusivamente con Acceleo no es recomendado. Por tanto, la implementación mencionada contiene sólo las funcionalidades básicas de una red de Petri y no contiene perfiles extra. Además, si se quisiera expandir a otros tipos de transformaciones sería complejo, ya que se trata de un único bloque de funcionalidad.

Análogamente, existe una implementación desarrollada por Pais, et al. [14] utilizando únicamente transformaciones ATL. En este papel los autores llegan a exponer en mayor detalle las complejidades de la implementación de la transformación ATL, algo que debido a la mayor extensión de este trabajo no es posible realizar. Un ejemplo de esta amplitud menor en el trabajo de los autores es la limitación a transformaciones a redes de Petri básicas, sin elementos adicionales como perfiles temporales.

4 Diseño

4.1 Preplanificacón

4.1.1 Resumen

A continuación, se detallarán las definiciones de los elementos de entrada al problema, sobre los que se operará durante el desarrollo del proyecto.

Además, se expondrán los detalles de la plataforma con la que se trabajará, Eclipse, y el conjunto de tecnologías integradas en Eclipse que se han escogido para abordar el problema, reduciendo la complexidad en la medida de lo posible, manteniendo todas las funcionalidades propuestas.

4.1.2 Eclipse

La plataforma Eclipse[9] es uno de los entornos de desarrollo integrado más populares entre los desarrolladores de software, tanto de producción como desarrollo, en múltiples lenguajes de programación.

Así mismo, Eclipse incorpora un sistema de gestión y herramientas de modelado, el Eclipse Modeling Framework (EMF[17]). El EMF utiliza un metamodelo estándar para procesar modelos, llamado Ecore, y un sistema de generación automática de código desde este mismo tipo de metamodelo.

Debido al alto número de usuarios utilizando la herramienta, se ha escogido como la plataforma sobre la que se desarrollará el proyecto, favoreciendo la expansión del código por otras partes y el uso de código ya desarrollado por los mismos. Igualmente, se hará uso de la herramienta EMF para el procesamiento de los modelos de entrada y salida.

Entre la multitud de distribuciones del entorno, se utilizará la más reciente, Eclipse IDE 2020-06.

4.2 Entrada del problema

4.2.1 UML2

El lenguaje de modelado unificado, o UML[21], es un lenguaje de modelado diseñado para la descripción y visualización de diseño de sistemas. UML es utilizado para numerosos modelos de representación de sistemas.

Uno de estos modelos de representación, la máquina de estados UML será la entrada de nuestra aplicación.

El estándar de diagramas de máquinas de estados utilizados será el estándar UML2, la implementación del estándar UML2.X para el Eclipse Modeling Framework. La versión del estándar en concreto será la 5.0, actual a día de creación del documento.

Por su accesibilidad, se podrá hacer uso fácilmente de más herramientas ya integradas en Eclipse.

4.2.2 MARTE

UML cuenta con puntos de expansión del metamodelo, llamados perfiles. Para el análisis de parámetros adicionales de la red de Petri generada, se utilizará un perfil de expansión de medidas temporales, en este caso, el perfil estándar MARTE[19].

MARTE es un conjunto de subperfiles, en los que se puede describir no sólo medidas temporales, sino planificación y reparto de recursos, coste de comunicaciones, y más. Ya que contiene todas las funcionalidades necesarias y es una de los perfiles más populares y completos para la descripción de modelos UML, lo he escogido como perfil de anotaciones adicionales.

En concreto se usarán los del perfil de análisis GQAM (Generic Quantitative Analysis Modeling), que engloba las prestaciones necesarias para describir las métricas en contexto de redes de Petri.

4.3 Elección de las Herramientas

4.3.1 Papyrus/MARTE

Papyrus[20] es un entorno de diseño de diagramas UML integrado en Eclipse. Papyrus está extensamente documentado y es cómodo e intuitivo de utilizar e instalar.

Papyrus cuenta con su propia perspectiva de Eclipse, adecuada para el diseño de diagramas, con numerosas ventanas y listados de propiedades de cada elemento UML del diagrama.

Adicionalmente, para añadir las funcionalidades de MARTE a la creación de los diagramas, existe una expansión del proyecto Papyrus en forma del proyecto Papyrus/MARTE. Papyrus MARTE permite añadir las anotaciones del perfil de MARTE a los diagramas desarrollados en Papyrus.

Cada diagrama viene descrito en diferentes vistas, todas englobadas en un único archivo con extensión .uml. El diseño del diagrama constituye la primera fase del proceso; una vez diseñado el diagrama, se podrá ejecutar la transformación completa en base al diagrama.

4.3.2 ATL

ATL[18] es una potente herramienta de transformación entre modelos, a través de esquemas de metamodelos en el esquema universal de modelos de Eclipse EMF.

ATL está compuesto por una nueva perspectiva, una serie de funciones relacionadas con la gestión de metamodelos, así como su propio lenguaje de programación y extensión de ficheros. El lenguaje define un conjunto de reglas de transformación entre dos metamodelos, un metamodelo de entrada, cuyo modelo de entrada debe cumplir, y un metamodelo de salida, que el modelo de salida cumplirá. La generación de los elementos del modelo de salida queda definida en las reglas de transformación, en función de las propiedades de los elementos del modelo de entrada.

Como primer paso en la transformación, se va a diseñar la conversión del metamodelo específico UML al metamodelo estándar de redes de Petri, PNML[16]. La mayoría del proceso de transformación toma lugar en ATL.

La transformación deberá generar cada elemento de la nueva red de Petri, así como convertir las distintas anotaciones MARTE a las propiedades intrínsecas de los elementos de la red.

4.3.3 Acceleo

Acceleo[15] es una herramienta de generación de archivos o código a partir de metamodelos de Eclipse. Acceleo toma un modelo conforme con el metamodelo Ecore, y produce una serie de ficheros de salida, cuyos contenidos quedan definidos por correspondientes plantillas de salida.

Para crear un archivo conforme al estándar PNML de gramática de redes de Petri, se ha decidido utilizar Acceleo, usando el archivo Ecore generado por la transformación ATL y creado una generación del archivo acorde con la gramática estándar PNML (.pnml).

La razón por la que se ha decidido usar Acceleo en lugar de otras herramientas de generación de archivos o código es por su excelente documentación, facilidad de uso e instalación y flexibilidad a la hora de generar los documentos y depurar el código.

4.3.4 GreatSPN Editor

Para la visualización de los modelos resultantes, y la evaluación de los mismos, se necesita una herramienta capaz de interpretar la salida de la generación Acceleo.

GreatSPN[6] es un *framework* de gestión y análisis de redes de Petri GSPN que lleva en desarrollo desde su primera iteración en 1985, escrita en Pascal. GreatSPN fue desarrollada por miembros de los departamentos de informática y telecomunicaciones de la universidad de Turín, y es actualmente la herramienta de gestión de redes GSPN más utilizada, y en los últimos 10 años ha recibido una aplicación que hace uso del *framework* GSPN para facilitar la interacción con el mismo.

La herramienta escogida es GreatSPN Editor, un programa de interpretación y simulación de redes de Petri y redes GSPN. Dentro del editor se puede visualizar la estructura gráfica de la red, modificar los atributos de los elementos, así como generar ejecuciones de simulación de la red, paso a paso y completas.

Será mediante estas simulaciones como se recogerán los valores de las métricas evaluadas durante el proceso de validación y análisis.

4.3.5 Eclipse Plugins

Eclipse tiene un sistema de integración de componentes a través de puntos de inserción. Estos componentes se llaman *plugins*. Estos componentes permiten expandir las funcionalidades del entorno de manera virtualmente ilimitada, ayudando al usuario a automatizar tareas, o realizar complejas operaciones tras interfaces accesibles.

Para crear una aplicación en Eclipse, se debe declarar como un *plugin*, tener sus valores identificativos como software empaquetado, y un punto de entrada al *plugin* (en este caso será un botón en la barra de herramientas) desde el cual comenzar a ejecutar las funcionalidades del *plugin*.

En este proyecto, se han unido varias aplicaciones de procesamiento de ficheros, y el objetivo es automatizar el proceso lo máximo posible sin necesidad de interacción por parte del usuario, en forma de modelo de caja negra.

5 Desarrollo

A la hora de la implementación, se han desarrollado las funcionalidades en proyectos de Eclipse diferentes, y enlazados en un único proyecto, el proyecto de *plugin* de la aplicación. A continuación, se detallará un flujo genérico de la aplicación, así como el desarrollo de los módulos individualmente.

5.1 Flujo de la aplicación

ATL

Tras la entrada de las rutas, la primera aplicación en ejecutarse será la transformación ATL, tomando el modelo UML de entrada y exportando un modelo de salida intermedio XMI. Para ello, la aplicación ATL estará empaquetada en un archivo .jar, con ejecución parametrizada, recibiendo las referencias y rutas de los elementos necesarios para la transformación, tanto los metamodelos, como el modelo de entrada, el directorio de salida y el archivo de transformación principal.

Acceleo

Una vez ejecutada la aplicación ATL, se ejecutará la aplicación Acceleo, tomando el archivo de salida de la transformación ATL y produciendo un archivo de proyecto GreatSPN en el directorio de salida. De manera similar, la aplicación Acceleo estará empaquetada en un archivo .jar, con ejecución parametrizada para recibir los datos ya determinados.

La estructura de salida del proyecto es un fichero de tipo .PNPRO, el formato de fichero de proyectos de GreatSPN Editor.

GreatSPN

La salida de la transformación Acceleo diseñada es ya en sí un proyecto GreatSPN, por lo que puede ser cargada posteriormente y analizada desde el programa de análisis de GreatSPN.

No obstante, se ha añadido al proceso del plugin el análisis a través de uno de los analizadores GreatSPN, MC_4CSL^{TA}[10]. El analizador $MC4CSL^{TA}$ está desarrollado por el mismo desarrollador de GreatSPN Editor, Amparore[24], y, al igual que GreatSPN Editor, está mayormente orientado a Linux. Por ello, se han utilizado los binarios de Linux de MC_4CSL^{TA} en la implementación. Para ejecutarlos desde la plataforma de Windows, se hace uso del Subsistema de Windows para Linux (WSL), introducido en Windows 10. Durante el desarrollo y testeo del proyecto desarrollado, se han utilizado las máquinas virtuales de las distribuciones Ubuntu y Debian para WSL.

La herramienta MC_4CSL^{TA} toma como entrada una red en formato GSPN (.net/.def). Para producir este tipo de archivo, se ha decidido hacer uso de parte del código de GreatSPN Editor, que ya tiene implementada una función de exportación de redes dentro de proyectos a formato GSPN. Para ello se ha utilizado una versión ligera de la aplicación compilada con esa única funcionalidad. Similarmente a las anteriores funcionalidades de las transformaciones, el exportador de redes es accedido a través de su

aplicación contenida en un archivo .jar, recibiendo el proyecto de GreatSPN Editor y el directorio donde exportar las redes como argumentos de ejecución.

Para llamar a la máquina virtual de Linux, se hará uso de la disponibilidad de ésta a través de la consola de comandos de Linux, mediante el comando `bash`, que carga la máquina virtual por defecto designada. Mediante la librería de `Runtime` se puede lanzar un proceso desde línea de comandos con las instrucciones necesarias, y leer la salida, tanto estándar como de error del proceso.

Un diagrama completo del flujo del proceso se puede encontrar en la Figura 5-1.

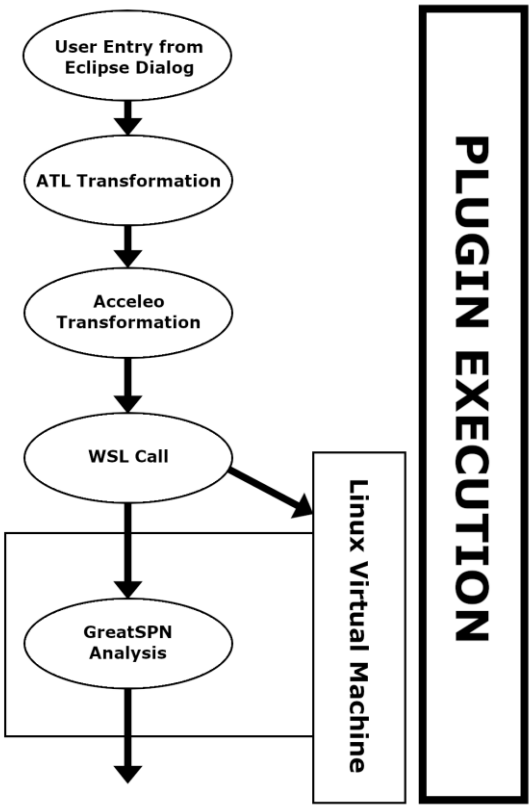

Figura 5-1: Flujo de ejecución del plugin

5.2 Transformación ATL

5.2.1 Arquitectura de ficheros

Para la primera parte de la transformación en ATL, se ha creado un proyecto ATL. En el proyecto, se hará uso de tres principales componentes:

Metamodelos

Para transformar un tipo de modelo a otro, se necesita las referencias de la estructura o metamodelos correspondientes. El archivo de entrada cumplirá el formato del metamodelo de entrada, y el modelo resultante cumplirá el formato del metamodelo de salida.

En este proyecto, el metamodelo de entrada será el estándar UML2 de Eclipse y el metamodelo de salida será el estándar PNML (Petri Net Markup Language)

Los metamodelos serán ambos archivos del formato de Eclipse Ecore.

Modelo de entrada

El modelo de entrada a transformar será un archivo de tipo .uml, conforme con el metamodelo de entrada UML2.

En este proyecto en particular, se ha generado el archivo con la herramienta de generación de modelos integrada en Eclipse, Papyrus.

Script de transformación principal

El conjunto de instrucciones de la transformación queda definido en un archivo de tipo .atl, donde se establecen los metamodelos de entrada y salida, y las transformaciones individuales de cada elemento relevante del modelo UML a elementos de PNML, así como la organización de la estructura interna de los mismos elementos. La arquitectura viene descrita también en la Figura 5-2.

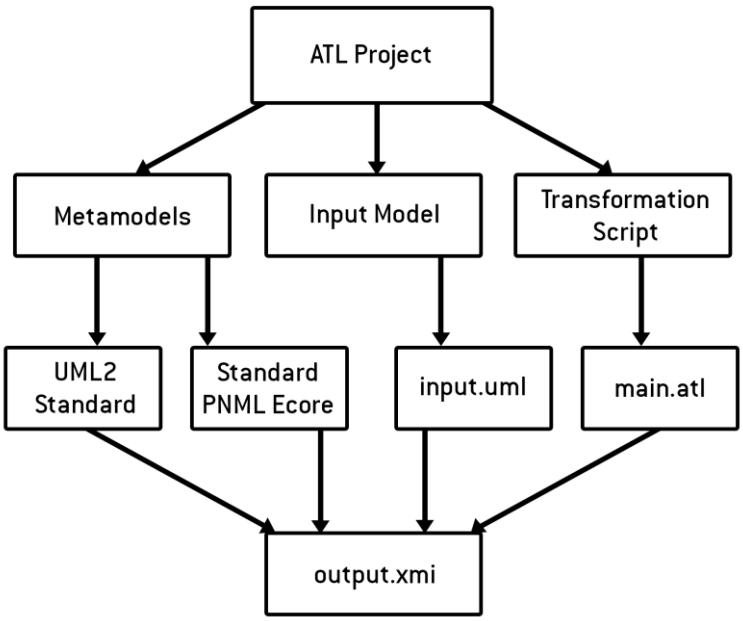

Figura 5-2: Arquitectura de ficheros ATL

5.2.2 Transformaciones de elementos

Cada elemento relevante del diagrama UML debe ser transformado a elementos correspondientes del metamodelo PNML. Los siguientes apartados describen las transformaciones llevadas a cabo por cada elemento de entrada UML relevante.

Máquina de Estados

La máquina de estados contiene un elemento `StateMachine`, simbolizando la máquina de estados global.

Para producir un modelo correcto del PNML, se necesita declarar un documento de `PetriNets`, así como la red global, representativa de la máquina de estados global de entrada. Ambos documentos son generados y enlazados con este elemento de entrada.

Regiones

En la transformación propuesta se unirán todas las regiones de la máquina de estados en una única región, o en equivalente en red de Petri, una única página.

Estados

Cada estado del diagrama simboliza uno o más lugares en la red de Petri. Para transformarlos, se descompondrá el estado en diferentes lugares, interconectados por arcos y transiciones.

El estándar UML2 declara 3 tipos principales de estados diferentes, `State`, `FinalState`, y `Pseudostate`, declarando las funcionalidades esperadas de estado convencional, estado final, y un conjunto de estados agrupados bajo el nombre de pseudoestado, incluyendo entre ellos estados iniciales, puntos de entrada y salida a submáquinas de estado y otros tipos de estado.

Mientras que `FinalState` es una subclase de `State`, y por tanto la conversión de estados finales es trivial habiendo implementado la de estados convencionales, este modelo de arquitectura de UML tiene la peculiaridad de que los estados iniciales requieren una transformación particular debido a que no comparten la mayoría de los atributos de los otros dos tipos de estados principales.

Para cada estado no inicial, se descompone en dos lugares, uno representando la entrada al estado, y otro la salida. Los dos lugares están conectados por la actividad interna del estado en forma de transición. En caso de no tener una actividad interna, la transición es siempre inmediata.

Para las tareas de salida o entrada al estado, su traducción son respectivos lugares situados antes y después de la transición de actividad interna respectivamente; y estarán conectadas con los lugares de entrada o salida al estado respectivamente.

Como los estados iniciales no tienen actividades internas o tareas de salida, son transformados a un único lugar.

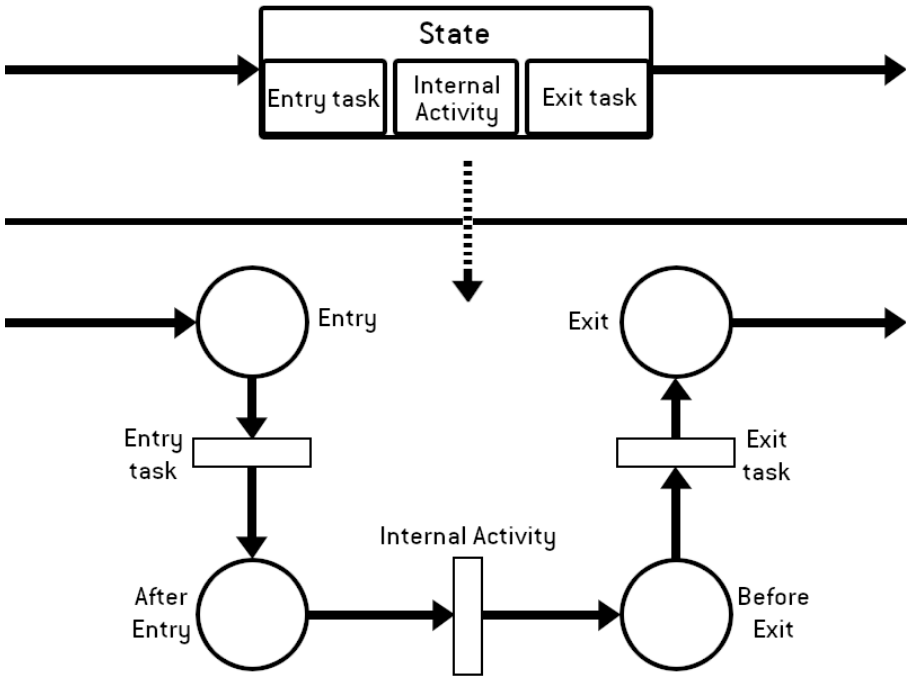

Figura 5-3: Transformación de estados

Como información adicional a los estados, puede haber anotaciones MARTE de tiempo de procesado para pasar por el estado, englobadas en el perfil `GaStep` de la librería de análisis, o la proporción de carga de elementos iniciales mediante el perfil `GaWorkload-Event`.

La proporción de carga es sencilla de transformar de un diagrama de máquina de estados a una red de Petri. En concreto en el entorno GreatSPN, el número de objetos inicial en cada lugar se marca con el atributo Marcado Inicial de los lugares.

La cantidad de tiempo de procesado en un estado se traduce como la duración de la transición interna o intraestado de cada lugar. Por ello, se adjudicará a la transición Internal Activity, descrita en la Figura 5-3, como el atributo `Delay` de la transición en el entorno GreatSPN.

Transiciones

Las transiciones procedentes de transiciones en la máquina de estados, o interestado, son representadas también como transiciones en la red de Petri.

Cada transición, tanto interestado como intraestado están conectados por arcos de lugar a transición y viceversa.

Aplicado a la transición, como comentario de MARTE, puede estar anotada la probabilidad de paso a través de la transición, por medio del perfil `GaStep`, de la librería de análisis. Esto se representa con el atributo `Weight` de las transiciones inmediatas en el entorno GreatSPN.

Por último, puede ser también aplicada una anotación de tiempo MARTE a una transición interestado. En ese caso, se trata como en el caso de las transiciones intraestado descritas en el apartado anterior.

Submáquinas de estados

En el estándar UML2, las submáquinas de estado son representadas contenidas en estados en la región externa. Para expandirlas en la página principal de la red, se tratará como una secuencia de estados y transiciones, conectada a la red principal por los puntos de entrada y salida de la submáquina, los cuales serán convertidos a lugares.

5.3 Transformaciones del perfil MARTE

La siguiente tabla muestra la correlación de los elementos del perfil MARTE pertenecientes al modelo UML de entrada a los parámetros temporales de la red de Petri de salida.

Anotación MARTE de entrada		**Parámetro de la red de Petri de salida**
Estereotipo	**Parámetro**	
`GaStep`	`hostDemand`	Tiempo (retraso) de la transición (interna para estados)
`GaStep`	`host`	Tiempo (retraso) de la transición (interna para estados)
`GaStep`	`prob`	Peso de la transición inmediata
`GaStep`	`rep`	Multiplicador de retraso del elemento
`GaWorkload-Event`	`pattern`	Número de objetos iniciales en un lugar
`GaScenario`	`respTime`	Tiempo (retraso) de la transición de cierre de la red

Tabla 1: Equivalencias de anotaciones MARTE

5.4 Generación de código Java del metamodelo

Para incorporar el metamodelo PNML en Acceleo, como paso intermedio, se ha creado un archivo de generación de código del Eclipse Modeling Framework (EMF) y generado el código Java correspondiente al metamodelo PNML.

5.5 Generación con Acceleo

5.5.1 Estructura de ficheros

Dado que sólo se genera un archivo de salida, la arquitectura de ficheros es bastante simplificada.

En este proyecto se han utilizado los ficheros básicos autogenerados por Acceleo, así como un servicio y un fichero de funciones auxiliares para gestionar la estructura espacial de los elementos en el diagrama.

5.5.2 Generación de fichero de salida

El fichero de salida generado por Acceleo será conforme con el estándar PNML, queda definido por la plantilla autogenerada al crear el proyecto, y sigue una estructura estándar de redes de Petri.

La estructura gráfica del fichero generado será en forma de una red de Petri circular, con los lugares ordenados en forma circular, y las transiciones en los puntos medios de los lugares que unen.

5.6 Proyecto Plugin

El proyecto consta de una interfaz de entrada a la aplicación en forma de botón en la interfaz de Eclipse. Una vez pulsado el botón, se debe desplegar toda la información que el usuario debe especificar para la realización de la operación en conjunto.

Con estos requisitos, se ha diseñado una ventana específica, construida con la herramienta de diálogos de Eclipse, con tres campos que el usuario debe rellenar la ruta del modelo de origen, presuntamente creado anteriormente con Papyrus/MARTE, la ruta al directorio de salida del proyecto GreatSPN, y la ruta a los binarios de Linux de MC_4CSL^{TA} (`DSPN-Tool`), que se usarán para el análisis de rendimiento de la red. La ventana en cuestión viene desplegada en la Figura 5-4.

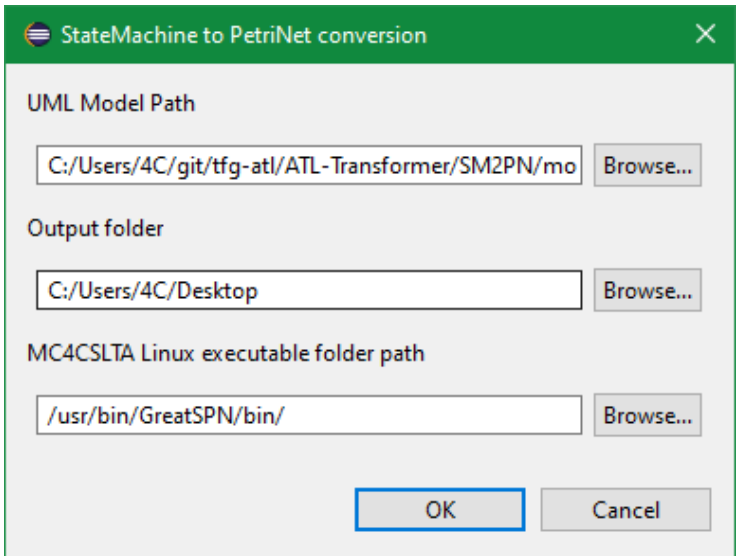

Figura 5-4: Ventana de entrada al plugin

Por último, si la ejecución se produjo exitosamente o surgió un error, se mostrará una ventana con los pasos y resultados del proceso. La ventana en cuestión viene desplegada en la Figura 5-5.

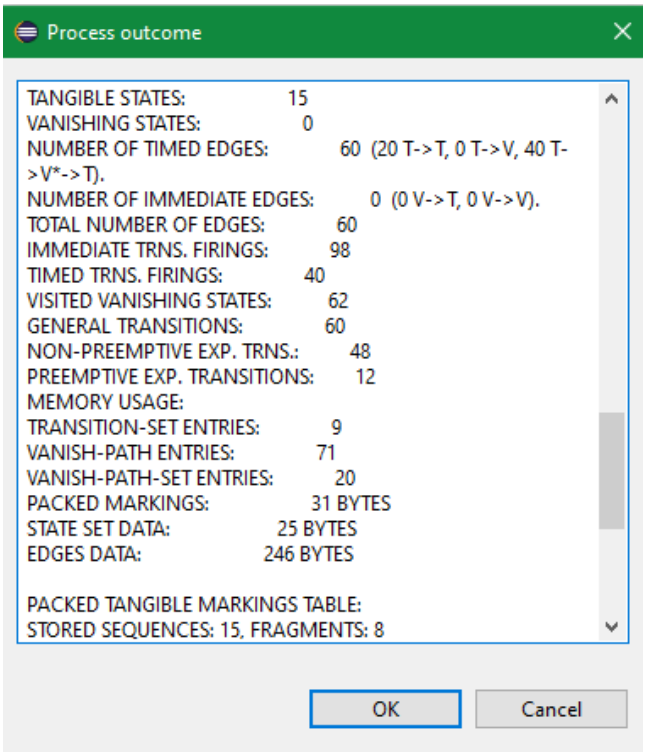

Figura 5-5: Ventana de resultados

6 Integración, pruebas y resultados

En este apartado se harán referencias a varias figuras del Anexo 0 como ejemplos de modelos de pruebas y validación.

Para validar el conjunto de transformaciones de modelos, se pueden visualizar los resultados desde GreatSPN Editor, ya que la salida de la transformación Acceleo es un archivo de proyecto de GreatSPN.

En la Figura 6-1 y Figura 6-2 se puede observar un ejemplo de transformación con una máquina de estados de prueba producida con Papyrus. La máquina contiene una serie de estados, un estado inicial, otro final, una serie de actividades, y una submáquina de estados contenida en el estado `ContainsSubMachine`.

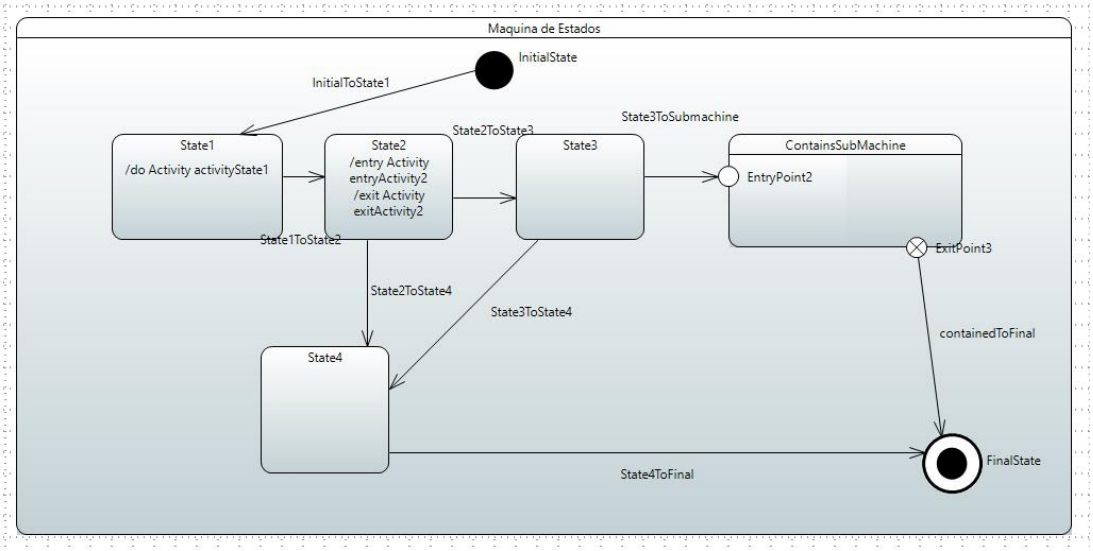

Figura 6-1: Ejemplo A, diagrama principal

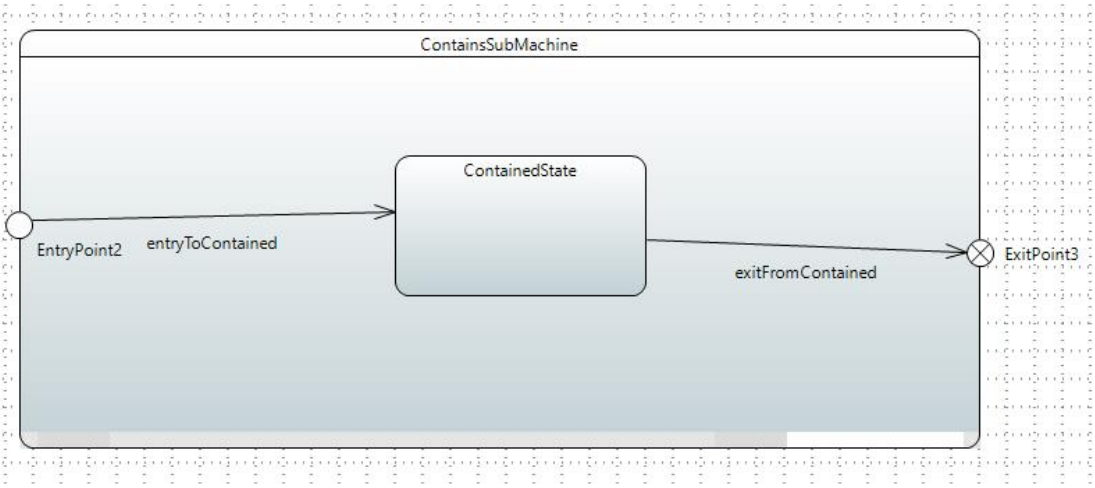

Figura 6-2: Ejemplo A, diagrama de la submáquina

El resultado de la transformación se encuentra en la Figura 6-3. En ella se puede observar el patrón circular a la hora de estructurar los lugares y transiciones descrito en el apartado 5.5.2, así como el despliegue de todos los elementos mencionados en el párrafo anterior.

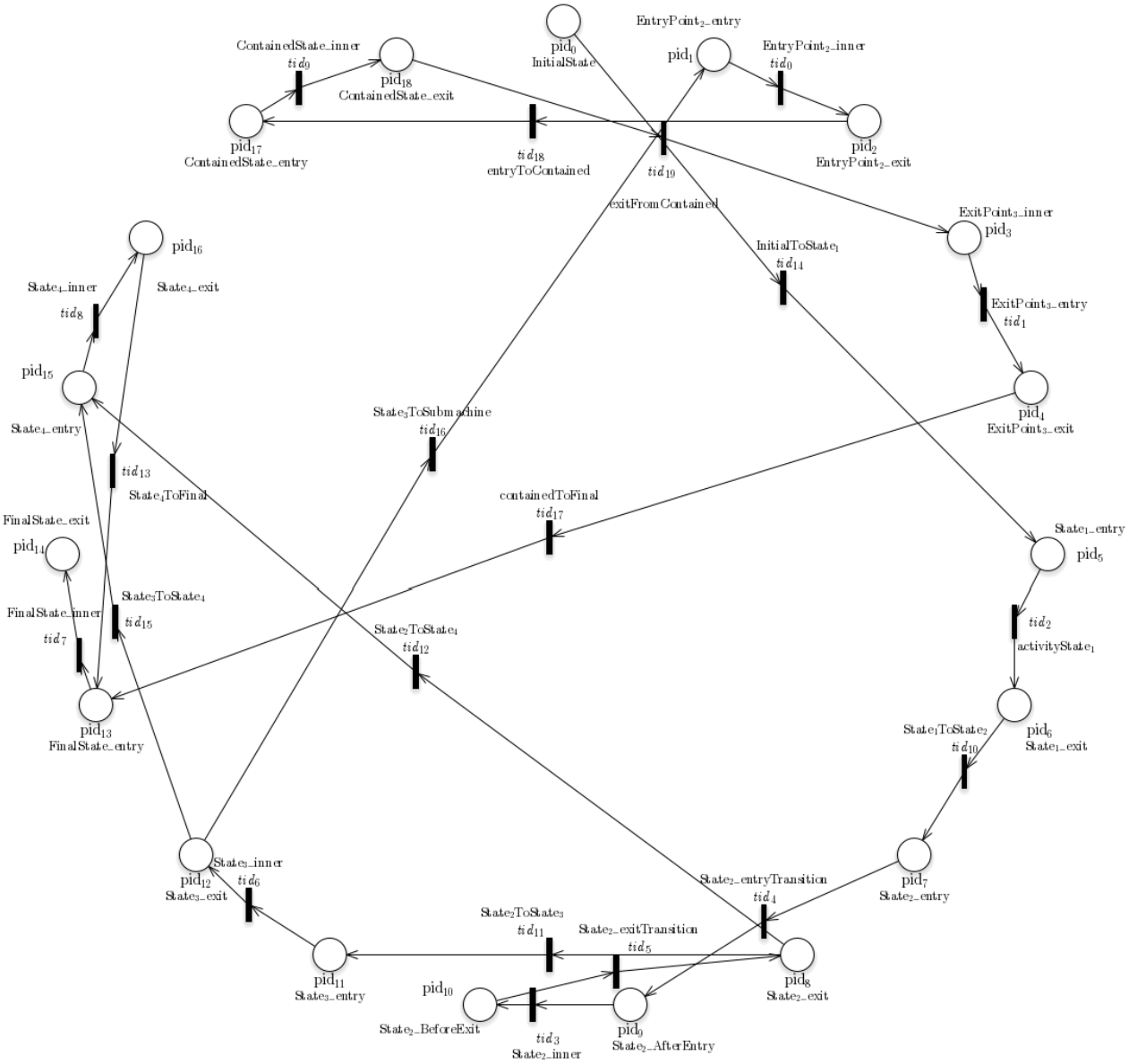

Figura 6-3: Ejemplo A, red de Petri generada

En la Figura 6-4 se muestra un ejemplo de diagrama Papyrus/Marte. En él hay cuatro anotaciones MARTE correspondientes a períodos temporales de espera (dos en estados, dos en transiciones), de 2.5 ms cada uno, así como una anotación MARTE correspondiente a 4 objetos iniciales en el estado Plasma.

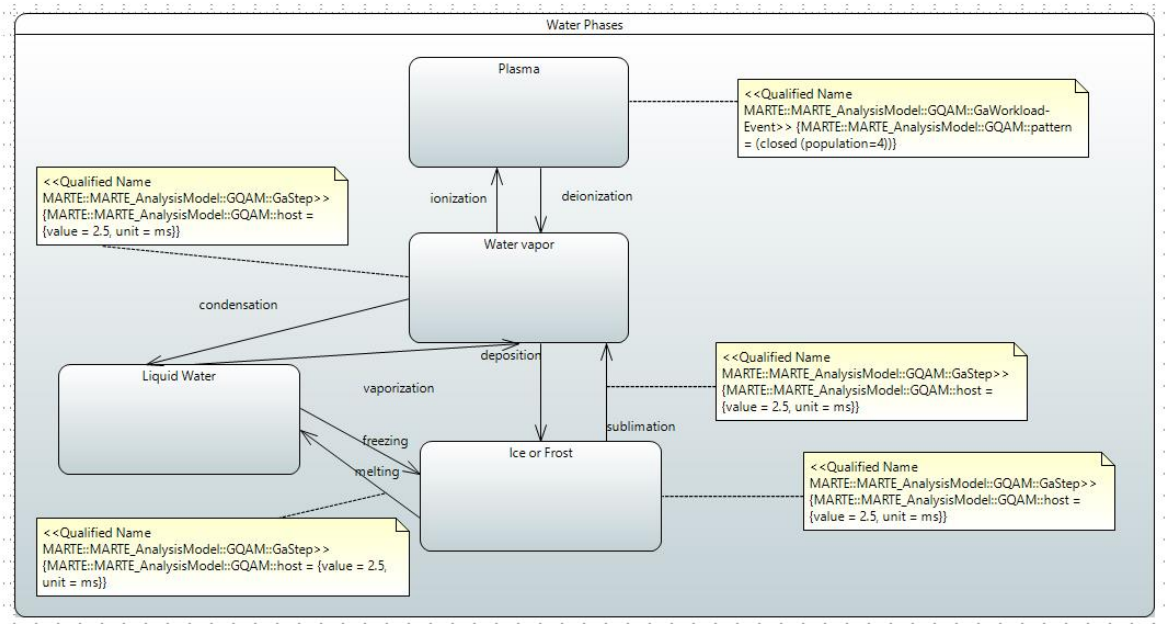

Figura 6-4: Ejemplo B, diagrama Papyrus/MARTE

En la Figura 6-5 se puede observar la red de Petri resultante tras la transformación. En ella se reflejan las cuatro transiciones temporizadas, representadas como rectángulos de mayor grosor en GreatSPN Editor, y el marcado inicial de 4 en el estado de `Plasma_entry`.

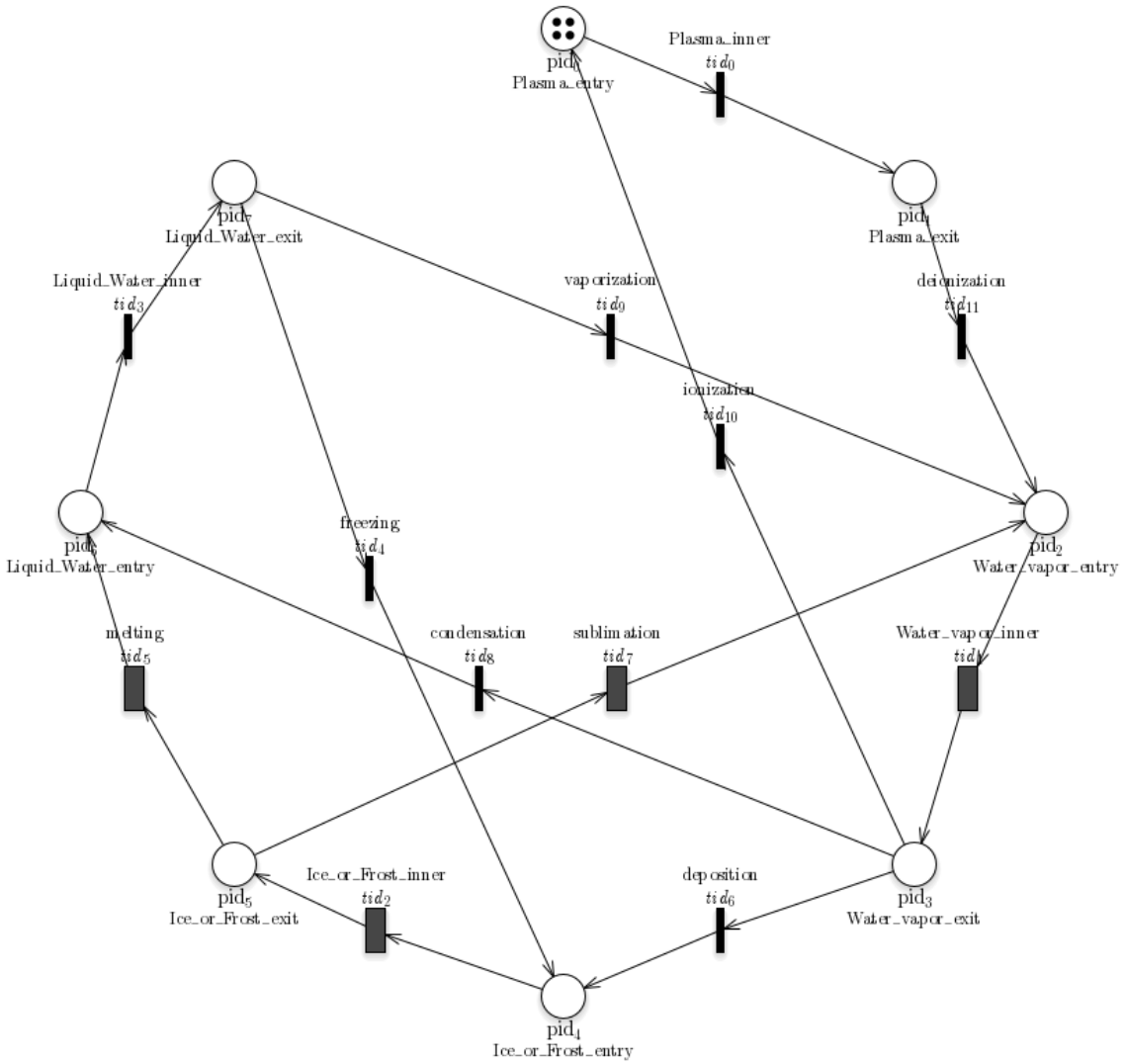

Figura 6-5: Ejemplo B, red de Petri temporizada generada

7 Conclusiones y trabajo futuro

7.1 Conclusiones

El trabajo desarrollado resulta en un análisis de modelos de máquinas de estados. Me resulta clara la utilidad de las redes de Petri para el análisis de comportamiento y rendimiento de sistemas modelados.

La herramienta de creación de *plugins* de Eclipse es excelente, ella permite el acceso a potentes funcionalidades a través de interfaces simples. Comenzar a crear *plugins* básicos en Eclipse es sencillo, pero la complejidad crece rápidamente con la cantidad de paquetes y librerías adicionales que son necesarios importar para añadir funcionalidades más avanzadas. Además, se requiere un entendimiento de la arquitectura software de Eclipse, ya que la serie de interfaces que utiliza para el manejo de ficheros está limitada en ciertas áreas, como el acceso a ficheros especializados, por ejemplo, la carga configuraciones de ejecución, requiriendo que formen parte del espacio de trabajo (*workspace*).

El entorno virtual WSL es una gran contribución a la plataforma, y la versatilidad a la hora de desarrollar software en Windows que proporciona es inmensa. La instalación del entorno es probablemente la más cómoda y sencilla de todas las posibles instalaciones de máquinas virtuales en el mercado, lo que permite tenerla configurada, funcionando, y cambiar de distribución a distribución distinta casi inmediatamente. Para usuarios que no actualizan frecuentemente su ordenador puede resultar tediosa la primera instalación debido al tamaño de las actualizaciones de Windows requeridas para el subsistema, pero la complejidad del proceso es ínfima.

No obstante, el *framework* de GreatSPN, así como la reciente aplicación GreatSPN Editor desarrollada para ese mismo *framework*, tienen muchas limitaciones de accesibilidad, los proyectos están pobremente mantenidos y su documentación es escasa. Varios enlaces de descarga a las versiones del programa (sorprendentemente incluida la actual) están rotos, y algunos paquetes de instalación del software están deprecados o son específicos a ciertas distribuciones.

Es una pena que el trabajo sobre este tipo de modelos sea mucho menos popular que otras tecnologías más comunes, porque seguramente se habrían desarrollado mejores herramientas para tratar y analizar redes de Petri. En el caso de que se desee operar extensivamente sobre redes de Petri recomendaría no usar el *framework* por sus limitaciones, pero sobre estudios menos específicos o extensos, la disponibilidad de la herramienta puede merecer la pena para evitar reconstruir otro *framework*.

7.2 Trabajo futuro

Existen numerosos frentes por los que explorar nuevos horizontes a partir de este proyecto.

Por una parte, encuentro interesante la premisa de reconstruir el *framework* GreatSPN y el editor. El *framework* lanzó su primera versión hace ya 35 años, y el editor hace 10, fue desarrollado por un estudiante, y ahora investigador, Amparore, de la universidad donde se desarrolló el *framework*, la universidad de Turín en Italia. El estado

actual del software es bastante precario y anticuado, y creo que sería beneficiario para futuros investigadores contar con un *framework* más robusto.

Sería un trabajo extenso reconstruir GreatSPN, ya que se trata de una herramienta multiplataforma, y con varias décadas y personal de desarrollo. Si se tratara de un proyecto en equipo y con financiación, yo mismo estaría grandemente interesado en el desafío.

Por otra parte, se puede explorar la idea de expandir el proyecto para otro tipo de transformaciones de modelos, como modelos de diagramas de secuencia, u otros modelos de descripción de funcionamientos de sistemas. En ese caso, recomiendo seguir un modelo de desarrollo similar al expuesto en este proyecto, si se desea implementar la funcionalidad en Eclipse. Existen numerosos modelos de descripción de ejecución o funcionamiento de sistemas, por lo que la estructura del proyecto es más importante que las transformaciones específicas.

Como recomendación adicional para implementar en Eclipse, recomiendo aprender las herramientas y de manejo de archivos y configuraciones de ejecución de Eclipse. En un futuro, es posible realizar la conexión entre las aplicaciones totalmente internamente dentro del plugin de Eclipse, sin necesidad de exportar las transformaciones como archivos .jar.

Por último, obviamente es posible realizar una implementación de esta aplicación en otros entornos de desarrollo que permitan una integración de nuevos componentes desarrollados como, por ejemplo, el popular entorno de desarrollo IntelliJ [25].

8 Referencias

[1] Jan Trowitzsch, Armin Zimmerman, “Using UML State Machines and Petri Nets for the Quantitative Investigation of ETCS”, 2006, Pisa, Italy.

[2] Kurt Jensen, “Coloured Petri Nets” 1992.

[3] John Anil Saldhana, Sol M. Shatz, “UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis”, Department of Electrical Engineering and Computer Science, University of Illinois at Chicago.

[4] John Anil Saldhana, Sol M. Shatz, Zhaoxia Hu, “Formalization of Object Behavior and Interactions from UML Models”, Department of Computer Science, University of Illinois at Chicago.

[5] Gino Wuytjens, “Transforming Statecharts to Place/Transition Petrinets”. University of Antwerp, Middelheimlaan 1, 2020 Antwerp, Belgium.

[6] Elvio Gilberto Amparore, Marco Beccuti, Gianfranco Balbo, Giuliana Franceschinis, Susana Donatelli, “30 Years of GreatSPN”

[7] Étienne André, Mohamed Mahdi Benmoussa, Christine Choppy, “Translating UML State Machines to Coloured Petri Nets Using Acceleo: A Report”. Universite Paris 13, Sorbonne Paris Cité, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France.

[8] Eclipse UML2 Standard: http://www.eclipse.org/uml2/5.0.0/UML

[9] Eclipse Website https://www.eclipse.org/

[10] MC_4CSL^{TA} Website http://www.di.unito.it/~amparore/mc4cslta/

[11] Murray Woodside, Greg Franks, Dorina C. Petriu, “The Future of Software Performance Engineering”. Carleton University, Ottawa, Canada.

[12] Murray Woodside, Dorina C. Petriu, José Merseguer, Dorin B. Petriu, Mohammad Alhaj, “Transformation challenges: from software models to performance models”

[13] Eric Simon, Kilian Stoffel, “State Machines and Petri Nets as a Formal Representation for Systems Life Cycle Management”. Université de Neuchâtel, Switzerland.

[14] Rui Pais, João Barros, Luís Gomes, “From SysML State Machines to Petri Nets Using ATL Transformations”. 5th Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS), Apr 2014, Costa de Caparica, Portugal. pp.227-236, ff10.1007/978-3-642-54734-8_26ff. ffhal-01274779f

[15] Acceleo Website https://www.eclipse.org/acceleo/

[16] Petri Net Markup Language Website http://www.pnml.org/

[17] Eclipse Modeling Framework Website https://www.eclipse.org/modeling/emf/

[18] ATL Transformation Language https://www.eclipse.org/atl/

[19] MARTE Real Time UML Profile https://www.omg.org/omgmarte/

[20] Eclipse Papyrus Website https://www.eclipse.org/papyrus/

[21] Unified Modeling Language (UML) https://www.uml.org/what-is-uml.htm

[22] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, "Modelling with Generalized Stochastic Petri Nets"

[23] Jorg Desel, Javier Esparza, "Free Choice Petri Nets". Cambridge University Press, 1995

[24] Elvio G. Amparore http://www.di.unito.it/~amparore/

[25] IntelliJ Website https://www.jetbrains.com/idea/

[26] GreatSPN Editor Source Code https://github.com/greatspn/SOURCES

[27] Elena Gómez-Martínez, José Merseguer, "ArgoSPE: Model-Based Software Performance Engineering". Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Spain.

[28] Vittorio Cortellessa, Raffaela Mirandola, "Deriving a queueing network based performance model from UML diagrams" Conference: Proceedings of the 2nd international workshop on Software and performance.

[29] Remenska D. et al. (2013) "From UML to Process Algebra and Back: An Automated Approach to Model-Checking Software Design Artifacts of Concurrent Systems." In: Brat G., Rungta N., Venet A. (eds) "NASA Formal Methods". NFM 2013. Lecture Notes in Computer Science, vol 7871. Springer, Berlin, Heidelberg

[30] Jiacun Wang, "Handbook of Finite State Based Models and Applications". October 2012. Chapman & Hall/CRC.

9 Glosario

FSM	Finite States Machine o Máquina de estados: Modelo de descripción de comportamiento de sistemas mediante una lista de configuraciones del sistema y transiciones entre ellas.
PN	Petri Net o Red de Petri: Modelo de descripción de comportamiento de sistemas de concurrencia mediante un grafo bipartido con los nodos representando configuraciones de elementos en el sistema, y procesos de paso entre uno o varios lugares.
SPE	Software Performance Engineering: Paradigma de desarrollo de software para el testeo y validación de otros sistemas.
MDE	Model Driven Engineering: Metodología de desarrollo de software a través de modelos intermedios entre el programador y el código generado para facilitar agilidad de desarrollo y compatibilidad entre sistemas.
EMF	Eclipse Modeling Framework: Framework y herramientas de generación de código de Eclipse para la programación mediante MDE.
UML	Unified Modeling Language: Lenguaje de diseño de modelado de sistemas estandarizado.
XMI	XML Metadata Interchange: Gramática estándar para el intercambio de información en formato XML.
ATL	Atlas Transformation Language: Herramienta de desarrollo de software a través de MDE para Eclipse que habilita generalizar transformaciones entre modelos a través de transformaciones generales entre sus metamodelos.
GSPN	Generalized Stochastic Petri Nets: Extensión de modelos de redes de Petri que añade la funcionalidad temporal al lenguaje.
PNML	Petri Net Markup Language: Gramática estándar en formato XML de modelos de redes de Petri.

WSL	Windows Subsystem for Linux: Máquina virtual nativa dentro del entorno de Windows 10 que ejecuta una distribución limitada de un sistema operativo Linux.
MARTE	Modeling and Analysis of Real Time and Embedded systems: Extensión o perfil de UML para añadir funcionalidades de representación de propiedades de tiempo real y gestión de recursos.
GQAM	Generic Quantitative Analysis Modeling: Submódulo del módulo de análisis de MARTE para definición de variables genéricas en el modelo.

10 Anexos

A Manual de instalación

Este anexo cubrirá los pasos para instalar la aplicación a una nueva instalación de Eclipse.

A.1 Eclipse

Todo el proyecto ha sido integrado en el popular entorno de desarrollo Eclipse. Existen numerosas distribuciones del entorno disponibles para descarga.

En este manual se expondrá el proceso de instalación para la distribución más común, *Eclipse for Java Developers*, y la versión actual a día de publicación de este trabajo, 2020-06. El entorno se puede descargar en la página oficial de Eclipse[9].

A.2 Plugin

El plugin se puede instalar como cualquier otro plugin local, a través de la opción *Help→Install New Software...→Add...→Archive...*, seleccionando el plugin comprimido en formato .zip.

Una vez instalado, basta con reiniciar Eclipse para tener acceso al plugin. El icono del botón del plugin, una unión de los logos de ATL y Acceleo, que aparecerá en la barra de herramientas de Eclipse, se encuentra mostrado en la Figura 10-1.

Figura 10-1: Icono del plugin

A.3 WSL y MC$_4$CSL^{TA}

Para ejecutar el analizador de modelos, es necesario tener una instalación de una distribución de Linux compatible instalada en el subsistema de Windows para Linux. Es posible que funcione con otras distribuciones, pero durante el desarrollo y validación del proyecto se han utilizado las distribuciones de Ubuntu 20.04 LTS y Debian para WSL.

Una vez instalada la distribución, se han de compilar los binarios de MC$_4$CSL^{TA}[26] con el comando "`make DSPN-Tool`". La ruta al binario resultante es la que se deberá introducir en la ventana de entrada del plugin para ejecutarse.

No obstante, se puede realizar una transformación independiente sin análisis sin tener acceso a MC$_4$CSL^{TA}.

B Manual del programador

El código del presente Trabajo Fin de Grado consta de 4 proyectos Eclipse. A continuación, se detallarán los contenidos y requerimientos de cada proyecto para poder modificar el código:

Proyecto `statetopetrinet.tfg.atl.transformer`

El proyecto statetopetrinet.tfg.atl.transformer cubre los archivos del módulo ATL, así como un módulo de ejecución de transformaciones en Java, para exportarlo como transformación independiente en un archivo .jar. Las dependencias requeridas son:

- `org.eclipse.m2m.atl`
- `org.eclipse.m2m.common`
- `org.eclipse.m2m.atl.emftvm`
- `org.eclipse.emf.ecore`
- `org.eclipse.emf.ecore.xmi`
- `org.eclipse.core.runtime`
- `org.apache.commons.cli`
- `org.eclipse.uml2.uml`
- `org.eclipse.uml2.uml.resources`
- `org.objectweb.asm`

El proyecto `statetopetrinet.tfg.atl.transformer` cubre los archivos del módulo ATL, así como un módulo de ejecución de transformaciones en Java, para exportarlo como transformación independiente en un archivo .jar.

Las transformaciones ATL se encuentran en las carpetas `resource` (metamodelos) y `SM2PM` (transformations). Dentro de la carpeta `src` se encuentra el código autogenerado del metamodelo pnml en el paquete `pnmlcoremodel` y el módulo de ejecución en Java en el paquete por defecto (`ATLRunner`).

Para modificar la transformación ATL, el fichero `SM2PM/transformations/main.atl` debe ser alterado. El fichero es una transformación ATL estándar, atendiendo sólo a los elementos relevantes a la transformación de máquinas de estados a redes de Petri, con enlazado de componentes al final de la transformación.

Si se desea transformar el metamodelo UML a otro metamodelo, se puede usar la estructura como plantilla cambiando el metamodelo de salida y generando el código del metamodelo de salida nuevo.

El lanzador de la transformación es una simple interfaz de lanzamiento. Recibe como argumentos el metamodelo de entrada, el modelo de entrada, salida y la transformación ATL en forma compilada .emftvm.

Para ejecutar la transformación dentro de Eclipse para tareas de depuramiento de la transformación, recomiendo ejecutarla con el lanzador `ATLRunner` o con una configuración de ejecución de transformación ATL EMFTVM en Eclipse.

Una vez terminada la edición del proyecto, se ha de exportar como archivo .jar con la configuración de ejecución `ATLRunner` para acceso desde el proyecto de plugin.

Proyecto `statetopetrinet.tfg.acceleo.transformer`

El proyecto `statetopetrinet.tfg.acceleo.transformer` cubre los archivos del módulo Acceleo, así como un módulo de ejecución de transformaciones en Java, para exportarlo como transformación independiente en un archivo .jar. Las dependencias requeridas son:

- `org.eclipse.acceleo.common`
- `org.eclipse.acceleo.model`
- `org.eclipse.acceleo.profiler`
- `org.eclipse.acceleo.engine`
- `org.eclipse.acceleo.parser`
- `org.eclipse.emf.ecore`
- `org.eclipse.emf.ecore.xmi`
- `org.eclipse.ocl`
- `org.eclipse.ocl.ecore`
- `org.eclipse.core.runtime`
- `com.google.guava`

La transformación tiene una única plantilla, `generate.mtl`, que produce el archivo .PNPRO. Esta transformación accede a funciones auxiliares en forma de los módulos `PNMLServices` y `AuxiliaryFunctions`, que realizan las opciones de generar el diseño estructural de los elementos y la adaptación de las anotaciones MARTE.

El archivo bin/.emtl generado tras modificar una plantilla .mtl debe ser copiado al directorio de el archivo .mtl para enlazarlo con el lanzador principal.

Acceleo tiene un lanzador propio muy accesible, el cual se ha usado para ejecutar la transformación completa en la clase Main, que recibe la ruta al modelo PNML y el archivo generate.emtl. Una vez terminada la edición del proyecto, se ha de exportar como archivo .jar con la configuración de ejecución Main para acceso desde el proyecto de plugin.

Proyecto `GreatSPNMiniExporter`

El proyecto `GreatSPNMiniExporter` no es más que una versión lite del código fuente de GreatSPN Editor. Su única función es cargar la principal red de un archivo de proyecto GreatSPN Editor como archivos de red GreatSPN (.net/.def). Las dependencias requeridas son:

- `org.eclipse.emf.ecore`
- `org.antlr.runtime`

El lanzador de la aplicación se encuentra en la clase `main.Main` y recibe la ruta del fichero .PNPRO, y se debe exportar como aplicación independiente en un archivo .jar.

Proyecto `StateToPetriNetPlugin`

El proyecto `StateToPetriNetPlugin` engloba la funcionalidad de la interfaz de la aplicación, así como el enlazamiento de los diferentes módulos en una única secuencia de ejecución. Las dependencias requeridas son:

- `org.eclipse.ui`
- `org.eclipse.core.runtime`
- `org.eclipse.core.resources`
- `org.debug.ui`
- `org.debug.core`
- `org.eclipse.jdt.launching`

El paquete `statetopetrinetplugin.dialogs` contiene los paneles y ventanas que muestra el plugin como entrada y salida del proceso, mientras que el código de ejecución del plugin viene definido en el gestor de comandos `LaunchHandler` en el paquete `statetopetrinetplugin.handler`.

La estructura de `LaunchHandler` extrae las rutas de entrada y salida de la ventana de diálogo de Eclipse, y ejecuta las aplicaciones en secuencia, las transformaciones y extracciones de los anteriores proyectos, y un análisis de la red resultante a través de WSL.

Los archivos de las transformaciones quedan localizados en el directorio `resources`. En él se encuentran los tres archivos .jar empaquetados de los proyectos anteriores, y en la carpeta `resources/atl` se encuentra la transformación ATL .emftvm así como el metamodelo PNML .ecore.

El *plugin* se puede exportar como cualquier otro proyecto de tipo *plugin* con un proyecto Eclipse de tipo *feature*.