UNIVERSIDAD NACIONAL DE RIO CUARTO

Trabajo de tesis para la obtención del grado de Licenciatura en Ciencias de la Computación

Máquinas de Estado UML como lenguaje de modelado del formalismo DEVS

Autor
Rodrigo Rubén Abella
rodrigoabella@gmail.com

Director de Tesis: *Mg. Ariel Gonzalez*

Rio Cuarto, Argentina
mes... 2015

Resumen

El desarrollo de sistemas dinámicos complejos requiere de estudios y análisis previos a la implementación, con el objetivo de detectar comportamientos no deseados. Por un lado, UML es un lenguaje ampliamente usado para el modelado de estos sistemas a través de máquinas de estados, entre otros mecanismos, pero carece de adecuadas herramientas de ejecución y simulación a fin de analizar el comportamiento real de los sistemas. Por otro lado, el formalismo *Discrete Event system Specification* (DEVS) viene a cubrir estas falencias y en la actualidad existen robustas herramientas que ejecutan y recopilan información de modelos DEVS. Este trabajo es una extensión de [1] y propone una implementación ad-hoc de modelado gráfico de sistemas dinámicos utilizando una simplificación de las máquinas de estados de UML (ME UML) las cuales fueron definidas en [2]. Además, se automatiza la transformación de las ME UML a modelos DEVS mediante una implementación en Java, con el objetivo de realizar simulaciones mediante herramientas disponibles, como PowerDEVS. Para ello, se desarrolla un editor de dichas máquinas de estados a través de un plugin para la plataforma eclipse. Además, se desarrolla en el mismo plugin una implementación en Java que permite ejecutar transformaciones QVT-Relations de ME UML en formato ecore a modelos DEVS en ecore tal como es definido en [1]. Allí, la transformación solo se puede realizar con la misma herramienta que define la transformación, MediniQVT.

Palabras clave: Máquinas de Estados, DEVS, UML, Eclipse.

Agradecimientos

En primer lugar, agradezco a mi familia quienes me han apoyado en todo este camino y que son parte de este logro.

Por otra parte, quiero agradecerle al tutor de la tesis al Mg. Ariel Gonzales de la facultad de Ciencias Exactas, Físico - Químicas y Naturales quien fue de gran ayuda para el avance de este trabajo, y por proponerme la realización del mismo.

List of Figures

Contents

Chapter 1

Introducción

En la actualidad, los proyectos complejos y a gran escala en diferentes áreas, requieren estudios previos a su implementación. La simulación y el modelado son técnicas aplicadas a dichos estudios previos con un fin específico, que es la de construir modelos del sistema para simular su comportamiento, analizar sus características, obtener información y elaborar conclusiones aplicables a la realidad.

Por otra parte, el desarrollo dirigido por modelos (Model-Driven Development, MDD) [3] es una metodología de desarrollo de software que se centra en la creación y explotación de modelos de dominio (es decir, representaciones abstractas de los conocimientos y actividades que rigen un dominio de aplicación particular). Tiene como objetivo aumentar la productividad mediante la maximización de la compatibilidad entre los sistemas (a través de la reutilización de modelos estandarizados), simplificando el proceso de diseño (a través de modelos de patrones de diseño que se repiten en el dominio de aplicación), y promoviendo la comunicación entre los individuos y equipos que trabajan en el sistema (a través de una estandarización de la terminología y las mejores prácticas utilizadas en el dominio de aplicación). Los modelos son desarrollados a través de una amplia comunicación entre los gerentes de producto, diseñadores, miembros del equipo de desarrollo y usuarios del dominio de la aplicación. Como conclusión, se obtienen modelos, que permiten el desarrollo de software y sistemas.

El Object Management Group (OMG) [4] tiene como iniciativa del MDD la arquitectura dirigida por modelos (Model Driven Architecture - MDA) [5] que es un acercamiento al diseño de software. Bajo la metodología MDA, la funcionalidad del sistema será definida en primer lugar como un modelo independiente de la plataforma (Platform-Independent Model o PIM) a través de un lenguaje específico para el dominio del que se trate. Dado un modelo de definición de la plataforma (Platform Definition Model o PDM), el modelo PIM puede traducirse entonces a uno o más modelos específicos de la plataforma (Platform-Specific Models o PSM) para la implementación correspondiente, usando diferentes lenguajes específicos del dominio, o lenguajes de propósito general como Java, C#, Python, etc. La traducción entre el PIM y los PSM se realizan normalmente utilizando herramientas automatizadas,

como herramientas de transformación de modelos (por ejemplo aquellas que cumplen con el nuevo estándar de OMG denominado Query/View/Transformation o QVT) [6]. Además, se propone la automatización de las transformaciones entre modelos y de la generación de código, pudiendo centrar el proceso de desarrollo de software en las tareas de modelado.

El modelo MDA está relacionado con múltiples normas, incluyendo el lenguaje Unified Modeling Language (UML) [7], que provee una notación gráfica y se ha convertido en el estándar para el modelado de diferentes aspectos de sistemas de software tanto en el ambiente académico como en desarrollos industriales. UML sigue el paradigma de orientación a objetos y permite la descripción de aspectos tanto estáticos como dinámicos de sistemas de software. Más que un lenguaje es un conjunto de lenguajes, en su mayoría notaciones gráficas, soportados por un número importante de herramientas propietarias y de código abierto. Si bien UML es uno de los medios preferidos de comunicación entre expertos del modelado, por su poder de representación gráfica, esta capacidad se encuentra acotada en términos de ejecución del modelo, es decir, la ejecución de una simulación.

Por consiguiente, en el campo de la simulación, el lenguaje de especificación de sistemas de eventos discretos (Discrete Event System Specification - DEVS) [8] [9] [2] es un formalismo modular y jerárquico para modelar y analizar sistemas de diversos tipos, en particular, sistemas de eventos discretos, sistemas de ecuaciones diferenciales (o sistemas continuos) y sistemas híbridos continuos y discretos. DEVS proporciona una base teórica del sistema para ejecutar modelos usando el protocolo de simulación DEVS. Sus modelos son de naturaleza jerárquica, y se componen de modelos atómicos y modelos acoplados con el fin de construir diseños con distintos niveles de abstracción.

UML es muy utilizado en la industria, y hoy en día existe una necesidad importante de disponer de herramientas de simulación de los modelos dinámicos de UML, a fin de analizar el comportamiento real de los sistemas complejos. Mas aún, es recomendable aplicar técnicas de modelado y simulación en etapas tempranas del desarrollo de software, dado que ayuda a detectar problemas poco visibles antes de su implementación. UML es muy poderoso en términos de su representación gráfica, pero débil en cuanto a la ejecución de sus modelos dinámicos.

Dentro de la comunidad de ingeniería de software, los diagramas de máquinas de estados son uno de los lenguajes de modelado de sistemas dinámicos de UML más utilizados. En el presente trabajo se propone unificar la potencia gráfica de UML partiendo de las maquinas de estados, con la potencia de simulación de DEVS hasta obtener un modelo de simulación para exportar a herramientas de simulación como PowerDevs, automatizando cada una de las transformaciones involucradas en el proceso anterior en una misma herramienta.

La propuesta presente, pretende abarcar todo el proceso de modelado, transformación y simulación en una misma herramienta basándose en los principales estándares del OMG, es por eso que se van a utilizar las herramientas mencionadas anteriormente para dicho proceso.

1.1 Contribución

La principal contribución de este trabajo es la construcción de un sistema que automatice el proceso de análisis, diseño y ejecución de máquinas de estados UML por medio del formalismo de simulación DEVS. Por un lado, se definen las representaciones (metamodelos) en formato Ecore de ambos lenguajes de modelado, y por otro lado se define la transformación para realizar el mapeo de máquinas de estados (compuestas) UML a máquinas de estados (simples) UML denominado proceso de "aplanado", a través de reglas de transformación en QVT-Relations. Luego se ejecuta una segunda transformación para mapear las máquinas de estados (simples) UML obtenidas a Modelos DEVS a través de reglas de transformación en QVT-Relations. Posteriormente el proceso realiza una transformación del modelo DEVS obtenido a código (M2T), basada en Acceleo. Finalmente se implementa un plugin para eclipse que automatiza todo el proceso de transformación con un editor de máquinas de estados UML ad-hoc permitiendo obtener luego de la transformación una implementación de un modelo DEVS para ser importado por la herramienta de Simulación **PoweDevs** [10] para su ejecución y análisis del modelo. Este proceso de transformaciones automatizada por el plugin se puede observar en la siguiente figura 1.1:

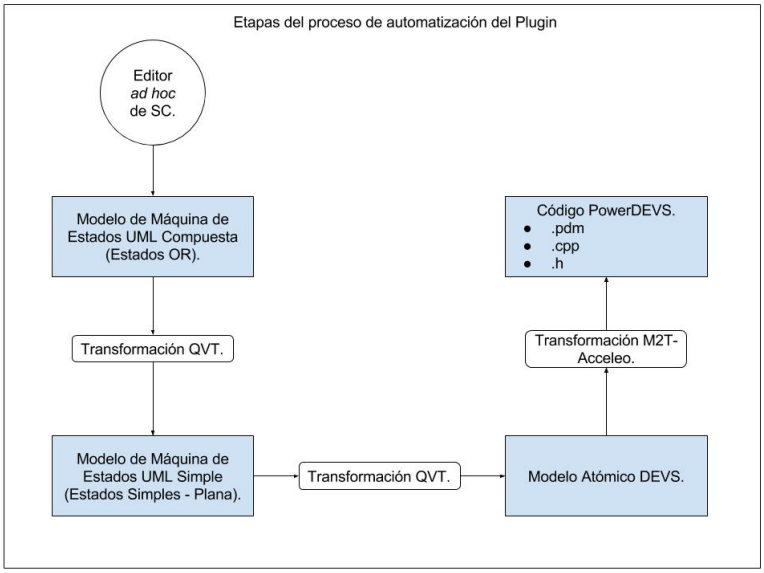

Figure 1.1: Etapas en el proceso de automatización.

1.2 Objetivos

Puntualizando el propósito de este trabajo de investigación, se define a continuación un conjunto de metas a alcanzar.

- Relacionar y vincular los conceptos subyacentes del modelado con UML y el modelado con el formalismo DEVS.
- Mostrar que modelos UML pueden ser transformados a modelos DEVS.
- Mostrar que la estructura y comportamiento de modelos DEVS pueden ser representados como modelos en formato Ecore, lo cual es necesario para aplicar las reglas de transformación en QVT-Relations.
- Implementar un editor de máquinas de estados UML ad-hoc.
- Presentar un mecanismo que permita ejecutar y analizar máquinas de estado UML a través del formalismo de simulación DEVS.
- Implementar un plugin para eclipse que permita realizar todo el mecanismo de transformación.

1.3 Organización del Trabajo

El trabajo es estructurado de la siguiente manera. En el capítulo 2, se hace mención de los trabajos relacionados con esta investigación, y puntualmente a la investigación conducida por Roque A. Cuello, Marcela C. Perez la cual fue utilizada como punto de partida para la actual iniciativa. En el capítulo 3, se introducen los distintos formalismos, metodologías, estándares y teoría general que dan la base a la problemática planteada, tales como el desarrollo de software basado en modelos (MDE, MDD, MDA), las máquinas de estados UML, y el formalismo DEVS. En el capítulo 4, se presenta el proceso de transformación de elementos estructurales y de comportamiento de las máquinas de estado (compuestas) UML a máquinas de estados (simples) UML y su posterior transformación a modelos DEVS. En este contexto se exhiben, el desarrollo de un metamodelo Ecore para modelos atómicos DEVS, la definición de un conjunto de reglas QVT-Relations que implementan el proceso de transformación. En la sección 4.4 se exhibe un ejemplo de aplicación de la propuesta. En el capitulo 5, se presenta la implementación realizada en Java para la automatización de las transformaciones. En el capitulo 6, se presenta el Plugin para Eclipse para dar soporte al proceso de diseño de modelos de ME UML y la necesidad de tener un editor ad-hoc.

Posteriormente, en el capítulo 7 se definen los métodos de pruebas utilizados. Finalmente, en el capítulo 8 a modo de conclusión, se discuten las ventajas y prestaciones que ofrece este proceso de transformación de modelos mediante reglas en QVT-Relations, en conjunto con la propuesta de continuación y trabajos a futuro que ofrece.

Chapter 2

Trabajos Relacionados

La transformación de componentes de UML a DEVS (y viceversa) no es una idea nueva, ni una iniciativa aislada. Integrar y unificar las comunidades que utilizan a uno u otro como método de modelado y simulación de sistemas, ha sido y es el objetivo principal de muchos trabajos de investigación que han dado luz a distintas metodologías, procesos e incluso la definición y desarrollo de frameworks. En este capítulo se mencionan algunos de los trabajos más influyentes para la propuesta, haciendo una aproximación a los métodos y soluciones que presentan, los cuales sirven de base general para ubicar e introducir la alternativa propuesta.

Una transformación de modelos consiste en el proceso de convertir un modelo de un sistema a otro, y para ello se requiere el modelo origen y destino, el metamodelo origen y destino respectivamente, y un conjunto de reglas de transformación. En la sección 2.1, se realiza una revisión de distintas propuestas y trabajos dentro del contexto de M&S orientados a la representación de un metamodelo para DEVS a través de distintas implementaciones.

Por otro lado, tomando esta iniciativa un paso más adelante, Mittal y Risco-Martín presentan eUDEVS, una metodología de transformación de Máquinas de Estados de UML a DEVS basada en XML, que incluye la definición de un metamodelo DEVS tanto como el conjunto de reglas de transformación, y además propone la inclusión de estos componentes en un proceso unificado DEVS, en un intento de proveer un framework que cierre la brecha entre UML y DEVS. Se analiza esta propuesta en la sección 2.2.

Por otra parte, Roque A. Cuello y Marcela C. Perez presentan en el paper Transformación Basada en Metamodelos de Máquinas de Estados UML a Modelos DEVS, una metodología de transformación de Máquinas de Estados de UML a DEVS definiendo transformaciones basadas en el estándar QVT (Query/View/Transformation) de la OMG. Se analiza esta propuesta en la sección 2.3, a modo de base teórica, motivación e inspiración para la alternativa de implementación que se presenta en este trabajo.

2.1 Propuestas en la comunidad M&S

Existen distintos enfoques para representar modelos DEVS, tal como Scalable Entity Structure Modeler con Complexity Measures (SESM/CM) [11]. Adecuado para el desarrollo de modelos jerárquicos basados en componentes, ofrece una base para modelar aspectos de comportamiento de modelos atómicos, provista por una especificación estructural y el almacenamiento del modelo usando XML, pero este enfoque es cercano a los expertos de la simulación más que a los expertos del dominio. Además, no hay razones para desarrollar explícitamente la máquina de estados atómica o su modelo de comportamiento. Esta es más una herramienta estructural y necesita mayor desarrollo para representar modelos atómicos usando XML.

En [12] y [13], otro metamodelo es definido para representar un sistema DEVS a través de XML. En ambos casos, JavaML [14] se utiliza para describir el comportamiento de un modelo atómico. Esta solución es buena para transformar modelos de plataforma específica (PSM) a modelos independientes de plataforma (PIM), pero no es una solución gráfica para modelar dichos sistemas. Dentro de este contexto, ATOM3 [15] es una buena solución. Tiene una capa de metamodelado que permite la especificación de lenguajes de modelado específicos del dominio, y una capa de modelado que soporta la construcción, modificación y transformación de modelos de dominio.

En el campo del modelado y simulación basado en UML, varios autores han abordado este tema desde diversas perspectivas. Choi [16] utiliza diagramas de secuencia UML para definir el comportamiento del sistema. En [17] , se introducen ocho pasos para construir modelos DEVS usando UML, pero en este caso se necesitan muchas decisiones humanas para transformar el modelo.

En [18] Zinoviev presenta un mapeo formal de DEVS a UML. Dentro de esta técnica, puertos de entrada y salida son mapeados a eventos UML. Variables de estado DEVS no continuas son mapeadas a estados UML, y aquellas que si son continuas son mapeadas a atributos de un estado UML. Además, Zinoviev emplea una combinación de eventos de tiempo especiales y eventos *after* para el manejo de transiciones internas. El mapeo presentado es elegante, sin embargo su representación en UML no tiende a proveer una representación unificada sobre el formalismo de modelado, con el propósito de composición, sino como un reemplazo de la especificación DEVS.

En [19], Huang y Sarjoughian presentan un mapeo para modelos acoplados en diagramas de estructura UML-RT, pero el uso de Perfiles UML para Planificabilidad, Rendimiento y Especificación Temporal (OMG 2005) es innecesario para el mapeo de DEVS a UML. Concluyen que UML-RT no es adecuado para el entorno de simulación, y afirman que el diseño de software y simulación son inherentemente distintos. En [20] , Borlang y Vanghuluwe desarrollan una metodología para transformar statecharts jerárquicos a DEVS.

En [21], proveen una transformación formal de autómatas temporizados de E/S en modelos de simulación DEVS. Sin embargo, el autómata temporizado es difícil de

comunicar y desarrollar el modelo de simulación.

En [22] se presenta un mapeo informal de DEVS a un statechart STATEMATE equivalente. Shulz indica que DEVS posee un mayor poder expresivo que los statecharts [23] y que cualquier modelo DEVS puede ser representado por un diagrama de actividades STATEMATE junto con una apropiada convención de identificadores para eventos.

En el contexto del paradigma MDA, Tolk y Muguira [24] muestran como las ideas complementarias y métodos de High Level Architecture (HLA) y DEVS pueden ser integrados en un dominio de aplicación M&S bien definido, dentro del framework MDS. HLA tiene arquitectura de simulación distribuida, independiente de la plataforma de computación. Provee una interfaz en la cual cada motor de simulación debe conformarse para participar en un ejercicio de simulación distribuida. Mientras es ampliamente utilizado en la industria de defensa, su adopción a la industria comercial ha sido restringida por la falta de poder de expresividad.

2.2 eUDEVS

En este paper, Saurabh Mittal (Dunip Technologies) y José L. Risco-Martín (Universidad Complutense de Madrid) presentan eUDEVS: Executable UML with DEVS Theory of Modeling and Simulation [25]. Allí, analizan no solo la especificación de la estructura y comportamiento de los modelos DEVS en UML, sino también un procedimiento de modelado de propósito general para modelos DEVS, que soporta la especificación, análisis, diseño, verificación y validación de una amplia variedad de sistemas basados en DEVS.

Muchos otros paradigmas como SES y modelado jerárquico DEVS pueden ser utilizados como interfaz con diagramas de estructura UML para proveer un modelo ejecutable. Sin embargo, se presenta un problema a nivel de modelos atómicos que contienen máquinas de estados finitas DEVS. Si bien la especificación de UML contiene diagramas de estados, el mapeo a la máquina de estados DEVS resulta en un incremento del mismo, con nueva información agregada para la cual no hay una especificación UML disponible, por ejemplo, tiempo de vida o timeouts de un estado, correspondiente al tiempo de avance en DEVS. Este problema ha sido remarcado por Mittal en [26] donde se presenta un argumento que indica que DEVS es más riguroso en situaciones de modelado de sistemas basados en estados.

El propósito principal de su trabajo apunta a especificar el lenguaje gráfico para que los ingenieros de sistemas puedan aprender cómo aplicar y utilizar UML para construir modelos DEVS, tanto estructura como comportamiento. Para ello, a nivel de diseño utilizan los componentes UML, paquetes y diagramas de clases. A nivel de comportamiento utilizan casos de uso UML, diagramas de secuencia, diagramas de tiempo y máquinas de estados.

En [27], Mittal propone un Esquema XML (W3C) para modelos acoplados DEVS que habilita al usuario a proveer una estructura de componentes de sistema en notación DEVS jerárquica. Esto describe la estructura y un comportamiento limitado para cualquier modelo DEVS. Sin embargo, no es una solución gráfica, porque el modelo debe ser escrito en XML para ser transformado al motor de simulación.

eUDEVS se basa en el comportamiento de dichos modelos, ya que la estructura del modelo ha sido profundamente estudiada y posee múltiples soluciones, como diagramas de bloques, diagramas de clase UML, etc. Se utiliza la metodología que maneja los Diagramas de Estados XML (SCXML) [28] como paso intermedio en el desarrollo de modelos atómicos DEVS desde máquinas de estados UML.

En él se propone una metodología de M&S basada en UML que consiste en 3 pasos. Primero, se sintetiza la estructura estática de la máquina de estados UML y se genera su correspondiente modelo SCXML. Segundo, se transforma el archivo SCXML a un modelo de máquina de estados DEVS finita y determinística (FD-DEVS SM, definida por Mittal en [27]), que especifica su comportamiento. En esta etapa el modelo es totalmente independiente de plataforma (PIM). Finalmente, a partir del modelo FD-DEVS XML se genera un modelo de simulación (PSM, específico de la plataforma) utilizando una serie de transformaciones basadas en XML, que puede ser ejecutado por el motor de simulación DEVS [13], [29], [27] (en particular, utilizan DEVSJAVA [30] como motor de simulación).

No solo demuestran que la transformación de modelos UML a componentes DEVS es posible, sino que también engloban estas técnicas de modelado, metamodelado y transformación, en un Proceso Unificado DEVS (DUNIP) que consiste en un proceso bifurcado para el modelado continuo de sistemas [31]. DUNIP permite el desarrollo de casos de prueba en paralelo con el desarrollo del modelo del sistema. Es una extensión lógica que permite a modelos UML, una vez transformados a modelos ejecutables, formar parte (como componentes) de un proceso unificado, así como lo es, por ejemplo, Rational Unified Process de IBM.

Finalmente, su investigación dio como resultado el desarrollo de un metamodelo para modelos ejecutables UML usando DEVS, una técnica de transformación de máquinas de estados UML a modelos DEVS usando XML, y la definición e inclusión de estos componentes y métodos en un framework unificado de modelado y simulación.

2.3 Transformación Basada en Metamodelos de Máquinas de Estados UML a Modelos DEVS

En este paper, Roque A. Cuello [32] y [33] y Marcela C. Perez definen un mecanismo que permita ejecutar y analizar máquinas de estados UML a través del formalismo de modelado y simulación (Modeling & Simulation - M&S) DEVS. En el trabajo definen formalmente la vinculación entre los elementos de las máquinas de estados UML y los

elementos del formalismo DEVS. Además, definen una representación (metamodelo) de DEVS y presentan un mapeo desde máquinas de estados UML a modelos DEVS, a través de reglas de transformación en el lenguaje QVT Relations (QVTR). Por último, construyen la implementación de un modelo DEVS que pueda ser importado por la herramienta de Simulación PowerDevs para la ejecución y análisis del modelo. Su investigación da como resultado un mecanismo que permite cerrar la brecha entre dos formalismos con distintas herramientas y tecnologías, distintas bases teóricas, pero unidos por un mismo propósito, que es el dar soluciones a problemas reales a través de la creación y el análisis de modelos abstractos, que sirve como punto de partida para la realización de este trabajo.

2.4 A Quick-Start Tutorial to Eclipse Plug-in Development

En este trabajo, Markus Belsiger presenta: A Quick-Start Tutorial to Eclipse Plug-in Development [33] una guía rápida para el desarrollo de un plugin en el IDE Eclipse orientada a los desarrolladores que quieren crear un plugin. En un principio se explica como es la creación de un proyecto básico y la estructura del mismo, luego establece como configurar el mismo para su ejecución y define que son los puntos de extensión que Eclipse brinda para los desarrolladores. Además explica como integrar y configurar en el plugin, la extensión que más interesa para el desarrollo de esta tesis, que son las interfaces de usuario, la que va a permitir interactuar con el usuario para iniciar el proceso de transformación de UML a DEVS.
Por último describe la diferentes alternativas para instalar un plugin en Eclipse explicando que son los sitios de actualización y como crearlos.

2.5 Flattening Statecharts without Explosions

En este paper, Andrzej Wasowski presenta: Flattening Statecharts without Explosions [34] basado en el aplanamiento de máquinas de estados UML, es decir transformar máquinas de estados compuestas a máquinas de estados simples.
En un principio define el concepto de "aplanar" para luego centrarse y explicar el problema de aplanar una máquina de estados.
Finalmente define un algoritmo para aplanar máquinas de estados, que sirve tanto para modelos planos y no planos lo que en la practica es muy bueno, y es una característica necesaria en el desarrollo de esta tesis. Además el algoritmo propuesto por Wasowski soporta estados compuestos concurrentes.

2.6 Refactoring Java code by transformation rules in QVT-Relation

En este trabajo [35], A. Gonzalez, M. Uva y M. Frutos definen un proceso de refactorización de código fuente de un programa en el lenguaje JAVA para optimizar el mismo

sin alterar su comportamiento. Para refactorizar el código realizan trasformaciones definiendo reglas en el lenguaje declarativo QVT-Relations, las cuales transforman un modelo java (código fuente de un programa java) a un modelo destino java (código fuente optimizado). Estos modelos se generan conforme a un metamodelo java que ellos definen.
Para realizar la transformación utilizan la herramienta MediniQVT que implementa la especificación QVT-R de la OMG en un motor QVT, la cual implica que los metamodelos y modelos sean escritos en formato Ecore.

Chapter 3

Nociones Preliminares

En este capítulo se exhiben los principales conceptos sobre las áreas de base que conforman este trabajo, estas son: las máquinas de estados, el desarrollo dirigido por modelos y el formalismo DEVS.

En la sección 3.1, se realiza una introducción a la Máquinas de Estados UML, a modo de definir en detalle sus componentes, comportamiento y propósitos, para dar la base del componente principal de nuestro trabajo. En la sección 3.2 se describen brevemente los principios de MDD, en particular la propuesta concreta de la OMG conocida como MDA. Y finalmente en la sección 3.3, se introduce la base teórica del formalismo DEVS dentro del contexto del Modelado y Simulación (M&S).

3.1 Máquinas de Estados

Hoy en día, existen necesidades de software cada vez más complejos y los tiempos de desarrollo son mucho mas cortos que años atrás, lo que implica tener que utilizar herramientas de diseño de alta productividad como lo son: sistemas de generación automática de código, optimizadores de código, lenguajes de alto nivel, librerías para la comunicación con periféricos, máquinas de estados, etc.

Las máquinas de estados poseen un nivel de abstracción más elevado que el de los lenguajes de programación, lo que facilitan el diseño del dominio de la aplicación. Proveen elementos gráficos que permiten describir con un grado de capacidad muy alta comportamientos complejos. A diferencia de las líneas de código, estos elementos son fáciles de comprender e interpretar y resultan muy adecuados para el intercambio de ideas, incluso con personas ajenas al proyecto además de facilitar la comunicación. Los modelos definidos son precisos y cada elemento tiene una semántica bien definida.

Un modelo de máquinas de estados no muestra el flujo de datos dentro del sistema, sino que muestra los estados del sistema y los eventos que provocan las transiciones de un estado a otro.

En UML una máquina de estados es utilizada para modelar los comportamientos de los objetos en un sistema, es decir modelar el aspecto dinámico del mismo. Son muy utilizadas para modelar sistemas de tiempo real y críticos, sistemas reactivos, circuitos, sistemas basados en protocolos. Las máquinas de estados son modelos de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores. Las máquinas de estados se definen como un conjunto de estados que sirve de intermediario en esta relación de entradas y salidas, haciendo que el historial de señales de entrada determine, para cada instante, un estado para la máquina, de forma tal que la salida depende únicamente del estado y las entradas actuales. En cada instante de tiempo la máquina se encuentra en un estado, y dependiendo de las entradas, actuales y pasadas, que provienen del ambiente, la máquina cambia, o no, de estado pudiendo realizar acciones que a su vez influyen en el ambiente. Las máquinas de estados tradicionales son una excelente herramienta para el tratamiento de problemas sencillos, pero su utilidad disminuye con problemas de moderada complejidad y resultan prácticamente inútiles en la descripción de sistemas complejos.

Los modelos de máquina de estados son una parte integral de los métodos de diseño propuesto por David Harel en la década de 1980 [36]. El método de Harel usa una notación denominada diagramas de estados (statecharts) que fue la base para la notación del modelado de máquina de estados en UML.

Los statecharts agregan a los diagramas de transición de estados, el anidamiento jerárquico de estados permitiendo modelar los sistemas con diferentes niveles de detalles, la concurrencia que es definida como ortogonalidad o paralelismo que permite tareas independientes entre si y ademas agregan la comunicación que hace posible que varias tareas se envíen mensajes y reaccionen ante un mismo evento fomentando la capacidad de moverse fácilmente entre diferentes niveles de abstracción.

Los SCs se incorporaron en las diferentes versiones de UML. A continuación se presentan los diferentes elementos y definiciones de los mismos basados en [37]. Cabe mencionar que también existen formalizaciones diferentes que se podrían tener en cuenta, como [38, 39].

Los SCs consisten esencialmente de estados y transiciones entre ellos. La principal característica de los SCs es que sus estados pueden refinarse, definiendo así una jerarquía de estados. La descomposición de un estado puede ser secuencial o paralela. En la primera, un estado se descompone en un autómata (estado `Or`), mientras que en la segunda se descompone en dos o más autómatas que se ejecutan concurrentemente (estado `And`). En la figura 3.1 el estado `Or` s0 es el estado de más alta jerarquía, que se descompone en 4 estados: s1, s2, s3 y s8. El estado s3, a su vez, se descompone paralelamente en los estados s4 y s7 (la descomposición paralela se indica con línea punteada).

Una *configuración de un SC* representa el estado actual del sistema en un instante dado [37]. Para describirla basta dar el conjunto de estados del SC en los cuales se encuentra el sistema. En el SC del ejemplo, las posibles configuraciones del sistema

son {s1}, {s2}, {s8}, {s3, s5, s9}, {s3, s6, s9}, {s3, s5, s10}, {s3, s6, s10}.

Más formalmente, siguiendo [37], se denomina S, TR, Π y A ($\Pi \subseteq A$) al conjunto de los estados, transiciones, eventos y acciones respectivamente de un SC. Asimismo, se define a un estado $s \in S$ o bien como un término básico de la forma $s = [n]$, como un término Or de la forma $s = [n, (s1, ..., sk), l, T]$ o como un término And de la forma $s = [n, (s1, ..., sk)]$, donde $nombre(s) =_{def} n$ es el nombre del estado. Para estados compuestos, $subestados(s) =_{def} (s1, ..., sk)$ es la secuencia de estados componentes de s, $inicial(s) =_{def} s1$ es el estado inicial de s, $T \subseteq TR$ es el conjunto de transiciones internas de s y l es el subestado activo de s. En la notación gráfica se etiqueta cada estado con su nombre, siendo las otras componentes deducibles del contexto.

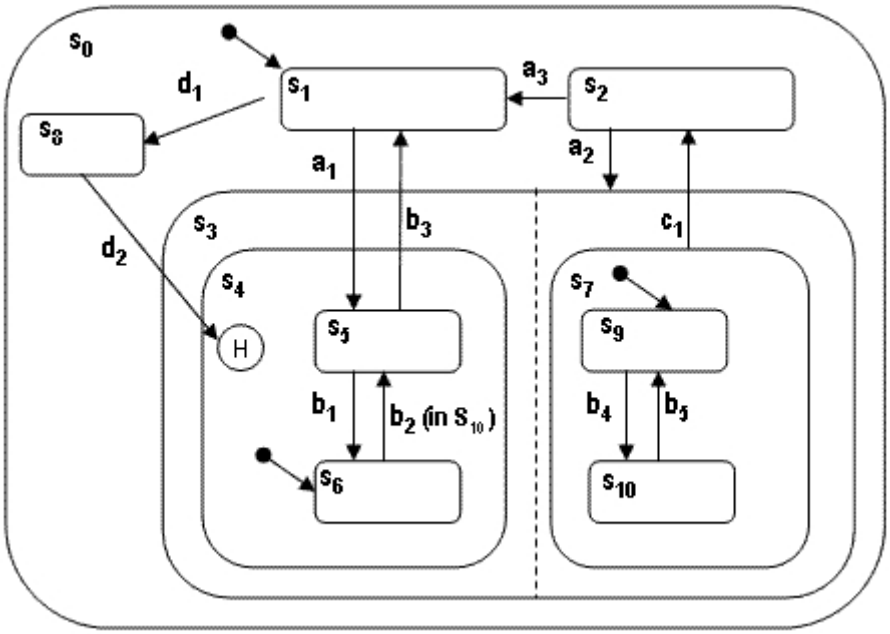

Figure 3.1: Representación gráfica de un SC.

Las transiciones son dirigidas y se representan como una tupla $t = (t', s_o, e, c, \alpha, s_d, ht)$, donde $nombre(t) =_{def} t'$ es el nombre de la transición, $origen(t) =_{def} s_o$ es el estado origen, $ev(t) =_{def} e$ es el evento que la "dispara", $cond(t) =_{def} c$ la condición de disparo, $acc(t) =_{def} \alpha$ es la secuencia de acciones que se realizan al efectuarse la transición, $dest(t) =_{def} s_d$ es el estado destino e $hist(t) =_{def} ht$ es el tipo de historia del estado destino (este concepto se explica más adelante).

La notación gráfica utilizada en las transiciones es $t : e, c/\alpha$, siendo las otras componentes de la transición deducibles del dibujo. Para que se pueda realizar una transición $t : e, c/\alpha$ con estado origen s_o y estado destino s_d es necesario que el sistema se en-

cuentre en una configuración a la cual pertenezca s_o, ocurra el evento e y la condición c se cumpla. La realización de la transición produce la ejecución de las acciones de α y el cambio de la configuración. Los SCs de UML 2.0 hacen una clasificación de los tipos de eventos internos o externos que pueden ocurrir: de señal (evento asíncrono), de llamada (solicitud de una operación), de cambio ($when(expr)$) y temporal ($alter(time)$). En el ejemplo de la figura 3.1 sólo se muestran los eventos asociados a cada transición, y en el resto de este artículo no se hacen distinciones entre los distintos tipos de eventos, ya que en la propuesta serán tratados por igual.

Cuando una transición tiene como destino un estado compuesto, debe aclararse cuál de los componentes de este estado es el destino de la misma. En el caso de un estado compuesto en forma paralela deben existir transiciones a cada uno de los componentes del mismo ($fork$). Para ello, la componente ht de una transición t es un elemento del conjunto $\{ninguna, profunda, superficial\}$, que indica a dónde llegarán las transiciones que tienen como destino un estado. El tipo de historia *ninguna* refiere al estado por defecto o inicial del estado compuesto. En el gráfico, este tipo de historia se indica representando la transición como una flecha al super-estado, y el estado por defecto se marca con una flecha hacia él sin origen (s6 y s9, en el ejemplo). Los otros dos tipos de historia indican que la transición tiene como destino el último estado visitado de un estado compuesto (aunque al inicio es el estado por defecto): $profunda$ indica que la transición entrará en el subestado más interno posible, y $superficial$ al del primer nivel. Para los tipos de historia $profunda$ y $superficial$ se utilizan en el gráfico las letras H* y H como destino de la transición, respectivamente. Finalmente, cuando una transición tiene como origen un estado compuesto paralelo y como destino a uno de nivel superior, la ejecución de esa transición lleva al sistema al estado destino, sin importar el estado en el que se encuentran las otras componentes del estado ($join$). En el ejemplo, estando el sistema en la configuración {s3, s5, s9} la transición b_3 lleva al sistema a {s1}.

3.1.1 Máquinas de Estados de UML 2.0

Las máquinas de estados de UML 2.0 son una extensión de los Statecharts definidos por Harel. El comportamiento de un objeto individual se modela mediante una máquina de estados. Una máquina de estados especifica las secuencias de estados por las que pasa un objeto a lo largo de su vida en respuesta a eventos, junto con sus respuestas a dichos eventos. La mayoría de las veces esto implica modelar la vida de las instancias de una clase, un caso de uso o un sistema completo. En UML son la mejor forma de especificar el comportamiento de los objetos reactivos, es decir, instancias que: deben responder a eventos externos o internos, tienen un comportamiento que depende de su historia, y tienen un ciclo de vida modelado como una progresión de estados, transiciones y eventos.

Una Máquina de Estados esta compuesta por:

- Un *estado*, es la condición o situación en la vida de un objeto durante la cual satisface alguna condición, realiza una actividad o espera un evento. Un objeto

permanece en un estado durante un tiempo finito.

- Un *evento* es la especificación de un acontecimiento significativo situado en el tiempo y espacio. En el contexto de las máquinas de estados, un evento es la aparición de un estímulo que puede disparar una transición de estados.
- Una *transición* es una relación entre dos estados que indica que un objeto que esté en el primer estado realiza ciertas acciones y entrará en el segundo estado cuando ocurra un evento especificado y se satisfagan unas condiciones especificadas.
- Una *actividad* es una ejecución no atómica en curso, dentro de una máquina de estados.
- Una *acción* es una computación atómica ejecutable que produce un cambio en el estado del modelo o devuelve un valor.

En UML 2.0 se utilizan las regiones para definir estados compuestos. Un estado que contiene exactamente una región identifica un estado simple de tipo Or. Un estado que contiene al menos 2 regiones identifica un estado ortogonal del tipo And. Las regiones no son visibles gráficamente en el modelo, dado que son sólo contenedores (container).

Estado

Es la condición o situación en la vida de un objeto durante la cual satisface alguna condición, realiza una actividad o espera un evento. Un objeto permanece en un estado durante un tiempo finito. Dentro de un estado se distinguen diferentes partes:

- *Nombre:* Texto que distingue un estado. Puede no contener nombre.
- *Efectos de Entrada/Salida:* Acciones ejecutadas al entrar y salir del estado.
- *Transiciones Internas:* Se manejan sin causar un cambio de estado.
- *Subestados:* Estructura anidada de un estado.
- *Eventos Diferidos:* Lista de eventos que no se manejan en ese estado sino que se posponen.

Evento

Un evento es la especificación de un acontecimiento significativo que tiene lugar en el tiempo y el espacio. Se pueden modelar cuatro clases de eventos:

- *De Señal:* Recepción de una comunicación asíncrona, explícita y con nombre, entre objetos. Un evento de Señal representa la acción de enviar (lanzar) de manera asincrónica un objeto para que otro lo reciba. Las excepciones son un tipo de señal interna más común. Tienen mucho en común con las clases, pueden contener instancias, atributos (parámetros) y operaciones, y también

pueden existir relaciones de generalización entre señales (jerarquías de señales). Son originadas por la acción de una transición de un estado a otro, el envío de un mensaje entre dos roles en una interacción o la ejecución de una operación. En UML 2.0 las señales se modelan como clases estereotipadas con <<signal>>.Las operaciones pueden enviar señales. Se indica con una dependencia y el estereotipo <<send>>. Las excepciones son otro tipo de señal, estereotipadas <<exception>>.

- *De Llamada:* Recepción, por un objeto, de una petición explícita sincrónica. Un evento de Llamada representa la invocación de una operación. Se sincronizan el emisor y el receptor. El emisor espera respuesta. Dos casos especiales son el evento de creación de un objeto (<<create>>) y el de su destrucción (<<destroy>>).

- *De Tiempo:* Llegada de un tiempo absoluto o transcurso de una cantidad relativa de tiempo. Un evento de Tiempo representa un instante en el tiempo mediante una expresión. La expresión puede ser:

  - Absoluta: at (<<temporal>>). Ejemplo: at (18:00hs).

  - Relativa: after (<<temporal>>). Debe usarse en el contexto de un disparador (trigger) de forma que el tiempo 0 es el de inicio de actividad del disparador. Por defecto el disparador es la entrada en el estado actual.

- *De Cambio:* Un cambio en el valor de una expresión booleana. Un evento de Cambio representa un cambio de estado o el cumplimiento de alguna condición (Notación: When (expresión booleana)). Ocurre una vez cuando el valor de la expresión cambia de falso a verdadero, no cuando pasa de verdadero a falso, y no se repite mientras la expresión sigue siendo cierta.

Un evento puede ser: sincrónico (Llamada) o asincrónico (Señales (excepciones), Paso de tiempo y Cambio de estado).

Transiciones

Es una relación entre dos estados que indica que un objeto que esté en el primer estado (origen) realiza ciertas acciones y entrará en el segundo (destino) estado cuando ocurra un evento especificado y se satisfagan unas condiciones especificadas.Cuando la transición es disparada el objeto pasa de un estado a otro.

Una transición se compone de:

- *Estado Origen:* Estado afectado por la transición.

- *Evento de Disparo:* Evento en el estado origen que provoca el disparo de la transición.
- *Condición de Guarda:* Expresión booleana que se evalúa cuando la transición se activa por el evento de disparo.
- *Efecto:* Comportamiento ejecutable como una acción que actúa sobre el objeto.
- *Estado Destino:* Estado activo tras completarse la transición.

3.1.2 Especificación de una Máquina de Estados

La especificación de una StateMachine es definida por la siguiente tupla:
SM = $\langle S, S_0, \Sigma, \delta, S_e \rangle$ donde:

- S: Es un conjunto de estados no vacíos.
- S_0 : Es el estado inicial.
- Σ: Es un conjunto finito de eventos.
- δ: S $\times\Sigma \rightarrow$ S es una función de transición.
- $S_e : S \rightarrow P(\Sigma)|()$ es una función que asigna a cada estado un conjunto de eventos de salida retardados.

3.2 Desarrollo Dirigido por Modelos

En esta sección se introducen los conceptos fundamentales relacionados con el desarrollo de software dirigido por modelos que se encuentran dentro del área *model-based engineering* (MBE). Se presentan las definiciones de MDE, MDD y MDA, centrándose en la propuesta *model-driven architecture* (MDA) de la OMG. Luego se hace una breve introducción al lenguaje de modelado UML, como así también al lenguaje de transformación de modelos QVT-Relations.

3.2.1 Ingeniería Dirigida por Modelos (MDE)

MDE (*model-driven engineering*) es un paradigma dentro de la ingeniería del software que apoya el uso de los modelos y las transformaciones entre ellos como piezas clave para dirigir todas las actividades relacionadas con la ingeniería del software.

MDE es un término mas amplio que MDD, ya que la ingeniería dirigida por modelos abarca todas las actividades de ingeniería de software, y no solo el desarrollo de sistemas.

De forma sencilla un modelo es una abstracción simplificada de un sistema o concepto del mundo real. Bajo esta definición podemos definirlo como: un modelo de un cierto $< X >$ es una especificación o descripción de ese $< X >$ desde un determinado

punto de vista, expresado en un lenguaje bien definido y con un propósito determinado. En donde <X> representa el objeto o sistema que se desea modelar, y puede ser algo concreto como abstracto. Es decir, el modelo proporciona una "especificación" de $< X >$ cuando se da la circunstancia de que $< X >$ no existe todavía (por ejemplo, es el sistema a construir), mientras que proporciona una "descripción" cuando representa un sistema que ya existe en la realidad y que queremos modelar con algún propósito determinado.

3.2.2 Desarrollo Dirigido por Modelos (MDD)

El desarrollo de software dirigido por modelos, conocido en inglés como *model-driven development* (MDD), es un paradigma de construcción de software cuyas motivaciones principales son la independencia de los desarrolladores de software a través de estandarizaciones y la portabilidad de los sistemas de software. El propósito de MDD es separar el diseño del sistema de la arquitectura de las tecnologías, para que puedan ser modificados independientemente. Para lograr esto, se asigna a los modelos un rol central y activo bajo el cual se derivan modelos que van desde los más abstractos a los concretos, este proceso se realiza a través de transformaciones sucesivas e iteraciones. Las transformaciones son el proceso en donde se toma un modelo de entrada y se produce otro modelo como salida. Las transformaciones son esenciales en el proceso de MDD.
La mayor importancia de este paradigma radica en que todo debe girar sobre la base de modelos, definidos a partir de metamodelos. El principal objetivo que cumple MDD es tratar de minimizar los costes y tiempo de desarrollo de las aplicaciones de software, en la búsqueda de mejorar la calidad, independiente de la plataforma donde el software se ejecuta, y garantice las inversiones en tecnología.

3.2.3 Arquitectura Dirigido por Modelos (MDA)

MDA (*model-driven architecture*) es la propuesta concreta de la OMG para implementar MDD, usando las notaciones, mecanismos y herramientas estándares definidos por esa organización.

MDA fue la primera de las propuestas de MDD, y la que consiguió hacer despegar el uso y adopción de los modelos como piezas clave del desarrollo de software.

MDA, al igual que MDD, aboga por la separación de la especificación de la funcionalidad de un sistema independientemente de su implementación en cualquier plataforma tecnológica concreta, y en el uso de transformaciones de modelos para transformar unos modelos en otros hasta obtener las implementaciones finales. Los estándares que la OMG ofrece para realizar MDA incluyen, entre otros:

- UML (Unified Modeling Language) como lenguaje de modelado.
- MOF (Meta-Object Facility) como lenguaje de metamodelado.
- OCL (Object Constraint Language) como lenguaje de restricciones y consulta de modelos.

- QVT (Query-View-Transformation) como lenguaje de transformación de modelos.
- XMI (XML metadata interchange) como lenguaje de intercambio de información.

La noción de metamodelo es fundamental en MDA. Un metamodelo es un modelo de un lenguaje de modelado. Un modelo es una representación de un sistema descripto en algún lenguaje bien definido. Una vista particular (o aspecto) de un sistema puede ser capturada por un modelo que es escrito en un lenguaje de su metamodelo.

La idea básica de MDA consiste en elaborar primero modelos de muy alto nivel, denominados modelos independientes de las plataformas o PIM (platform-independent models) y completamente independientes de las aplicaciones informáticas que los implementarán o las tecnologías usadas para desarrollarlos o implementarlos. MDA define entonces algunos procesos para ir refinando esos modelos, particularizándolos progresivamente en modelos específicos de la plataforma a usar o PSM (platform-specific models) cada vez más concretos conforme se van determinando los detalles finales.

La figura 3.2 muestra el contexto de cada uno de los términos descriptos en este capítulo.

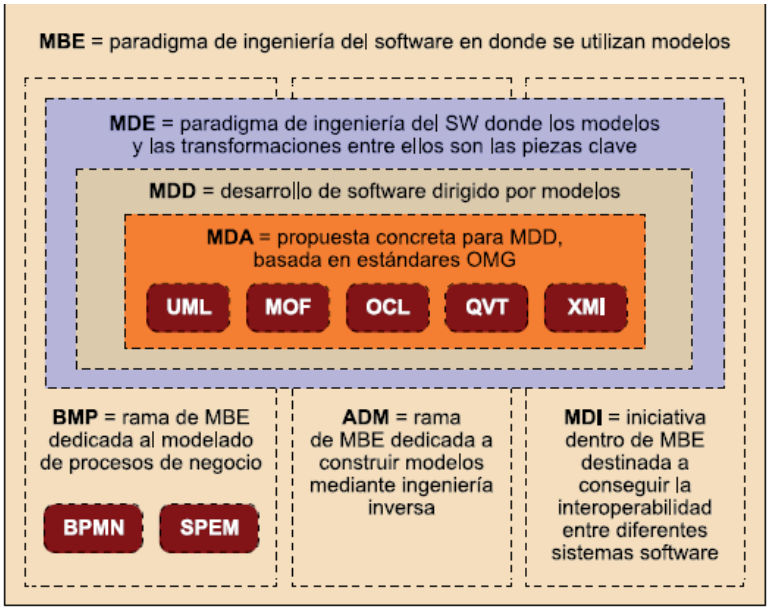

Figure 3.2: Relación entre los contextos de MDE, MDD y MDA.

3.2.4 Lenguaje Unificado de Modelado (UML)

Es un lenguaje estándar para visualizar, especificar, construir y documentar los artefactos de un sistema de software.

En 1997 UML fue adoptado como un estándar por el OMG (Object Management Group). UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

UML proporciona un vocabulario y reglas para permitir una comunicación. Se centra en la representación gráfica de un sistema. Este lenguaje nos indica cómo crear y leer los modelos, pero no dice cómo crearlos. Esto último es el objetivo de las metodologías de desarrollo.

Las funciones básicas de UML son:

- Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Un modelo UML esta compuesto por tres clases de bloques de construcción:

- Elementos: Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
- Relaciones: relacionan los elementos entre sí.
- Diagramas: Son colecciones de elementos con sus relaciones.

Un diagrama es la representación gráfica de un conjunto de elementos con sus relaciones. En concreto, un diagrama ofrece una vista del sistema a modelar. Para poder representar correctamente un sistema, UML ofrece una amplia variedad de diagramas para visualizar el sistema desde varias perspectivas. UML incluye los siguientes diagramas:

- Diagrama de casos de uso.
- Diagrama de clases.
- Diagrama de objetos.

- Diagrama de secuencia.
- Diagrama de colaboración.
- Diagrama de estados.
- Diagrama de actividades.
- Diagrama de componentes.
- Diagrama de despliegue.

Además, UML permite el modelado de comportamientos dinámicos de un sistema, utilizando Máquinas de Estados.

3.2.5 Transformación de Modelos

Una transformación de modelos es el proceso de convertir un modelo de un sistema en otro modelo. Una transformación establece un conjunto de reglas que describen cómo un modelo expresado en un lenguaje origen puede ser transformado en un modelo en un lenguaje destino. La figura 3.3 describe los elementos incluidos en una transformación de modelos: un modelo destino B es obtenido desde un modelo origen A utilizando un motor que ejecuta la especificación de una transformación. El modelo origen A conforma al metamodelo MA, lo mismo ocurre con el modelo destino B respecto a su metamodelo MB. Similarmente una especificación de una transformación debe conformar a un metamodelo. En resumen, una transformación requiere:

1. modelos origen y destino,
2. metamodelos origen y destino y,
3. la especificación de la transformación.

La OMG identifica, a través de MDA, cuatro tipos de transformaciones de modelo a modelo (M2M) dentro del ciclo de vida del software:

1. *Transformaciones PIM2PIM*: son utilizadas para el tratamiento de modelos independientes de la plataforma y se aplican cuando se obtienen, filtran o especializan unos PIM a partir de otros.
2. *Transformaciones PIM2PSM*: transforman un modelo independiente de la plataforma en su correspondiente modelo en una plataforma concreta.
3. *Transformaciones PSM2PSM*: se utilizan para refinar un modelo desplegado en una plataforma concreta.
4. *TransformacionesPSM2PIM*: abstraen modelos desplegados en plataformas concretas en modelos independientes de la plataforma.

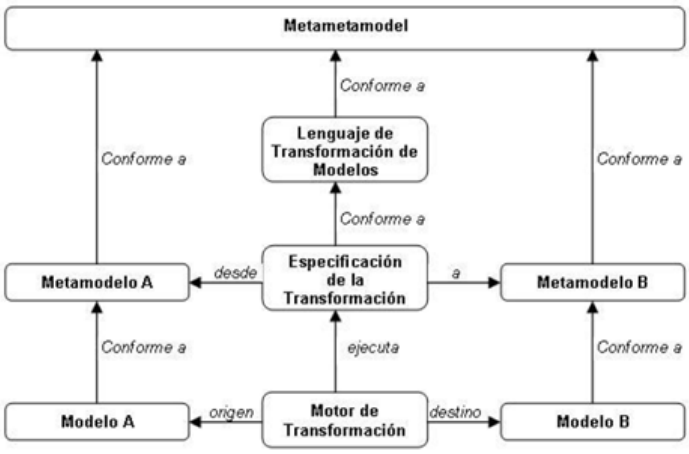

Figure 3.3: Esquema de transformación de modelos.

3.2.6 QVT

QVT (del inglés, query-view-transformation) es la propuesta del OMG para resolver el problema de la transformación de modelos. Se trata de un estándar para la definición de transformaciones sobre modelos MOF.

La especificación de QVT depende de otros dos estándares de la OMG como son MOF y OCL. De esta manera, la utilización de la especificación de QVT para especificar transformaciones, aporta reutilización de tecnología que sigue estándares y reducción de la curva de aprendizaje de la herramienta. La dimensión del lenguaje define los diferentes lenguajes de transformación presentes en la especificación QVT. Concretamente son tres: Relations, Core y Operational, y la principal diferencia entre ellos es su naturaleza declarativa o imperativa. Estos lenguajes se organizan en una estructura por niveles que se muestra en la figura 3.4:

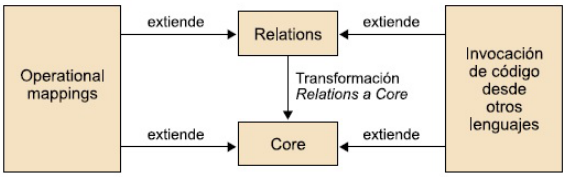

Figure 3.4: Estructura

3.3 Formalismo DEVS

Durante las últimas décadas, la rápida evolución de la tecnología ha producido una proliferación de nuevos sistemas dinámicos, generalmente hechos por el hombre y de gran complejidad. Ejemplos de ellos son las redes de computadoras, sistemas de producción automatizados, de control de tráfico aéreo; y sistemas en general de comando, de control, de comunicaciones y de información. Todas las actividades en estos sistemas se deben a la ocurrencia asincrónica de eventos discretos, algunos controlados (tales como el pulsado de una tecla) y otros no (como la falla espontánea de un equipo). Esta característica es la lleva a definir el término de Sistemas de Eventos Discretos.

Las herramientas matemáticas que hoy disponemos (básicamente ecuaciones diferenciales y en diferencias) fueron desarrolladas durante los últimos doscientos años para modelar y analizar los procesos conducidos por el tiempo que generalmente uno encuentra en la naturaleza. El proceso de adaptar estas herramientas y desarrollar nuevas para los sistemas conducidos por eventos tiene solo unos pocos años. Por este motivo, encontramos en la teoría de los sistemas de eventos discretos no sólo una serie de herramientas específicas para atacar problemas de modelización, simulación y análisis de sistemas altamente ligados a la práctica de la ingeniería y a los problemas de la informática, sino también un campo fértil para el desarrollo de nuevas técnicas y teorías debido a la cantidad de problemas aún abiertos en el área. Dentro de los formalismos mas populares de representación de sistemas de eventos discretos (DES) están las Redes de Petri, las Statecharts, Grafcet, Grafos de Eventos y muchas generalizaciones y particularizaciones de los mismos. Con respecto a las herramientas de análisis, sin dudas las más interesantes son las obtenidas con la introducción de estructuras algebraicas de tipo dioides max-plus y min-plus.

Orientado a los problemas de modelización y simulación de DES, en la década del 70, el matemático Bernard Zeigler propuso un formalismo general para la representación de dichos sistemas [8]. Este formalismo, denominado DEVS (Discrete Event System specification), es de hecho el formalismo más general para el tratamiento de DES. El hecho de estar fundado en la base de la teoría de sistemas, lo convierte en un formalismo universal, y por lo tanto, todos los otros formalismos mencionados en el párrafo anterior pueden ser absorbidos por DEVS (es decir, todos los modelos representables en dichos formalismos pueden ser representados en DEVS).

DEVS es un formalismo modular y jerárquico para modelar y analizar sistemas de diversos tipos. En particular, sistemas de eventos discretos, sistemas de ecuaciones diferenciales (o sistemas continuos) y sistemas híbridos continuos y discretos.

3.3.1 Segmentos de Eventos

Un segmento de eventos se define formalmente de la siguiente forma:
Sea w: (t0, tn) $\Longrightarrow$ A $U\{\varnothing\}$ un segmento sobre una base de tiempo continua (o sea, una función de t perteneciente al intervalo (t0, tn) de los reales) y el conjunto A $U\{\varnothing\}$.

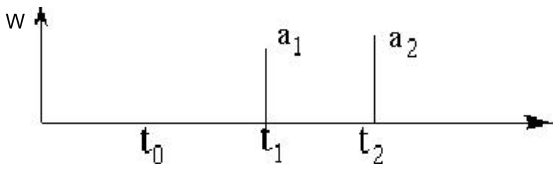

Figure 3.5: Un segmento de Eventos

Aquí ∅ es un elemento que no pertenece a A y representa la ausencia de evento. Luego w es un segmento de eventos si existe un conjunto finito de puntos t1, t2, t3, ..., tn-1 ∈ (t0, tn) tales que w(ti) = ai ∈ A para i = 1, ..., n-1, y w(t) = ∅ para todo otro t ∈ (t0, tn).

3.3.2 Definición del Formalismo DEVS

Un modelo DEVS atómico clásico es una estructura:

$$\mathrm{M} = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$$

donde:

- X es el conjunto de valores de entrada
- Y es el conjunto de valores de salida
- S es el conjunto de estados
- $\delta_{int} : S \to S$ es la función de *transición interna*
- $\delta_{ext} : Q \times X! \to S$ es la función de *transición externa*, siendo $Q = \{(s, e) \,|\, s \in S, 0 \leq e \leq ta\,(s)\}$ el conjunto de estado total e el *tiempo transcurrido* desde la última transición.
- $\lambda : S \to Y$es la función de salida
- $ta : S \to \Re^{+}_{0,\infty}$ es la función de *avance de tiempo*

Se presenta la figura 3.6 para describir el significado de los elementos de la estructura definida.

Un sistema que está en un estado $s \in S$, permanecerá en el mismo por un tiempo $ta(s)$ salvo que, habiendo transcurrido un tiempo e menor o igual que $ta(s)$ en ese estado, ocurra un evento de entrada con valor $x \in X$. En tal caso, el sistema experimentará una *transición externa* cambiando al estado $s' = \delta_{ext}\,(s, e, x)$. En la figura esto está ilustrado por medio de la flecha continua que va de s a s' .

En cambio, si el tiempo transcurrido e es igual a $ta(s)$ sin que se hayan producido eventos externos, un evento interno ocurrirá dando lugar a una *transición interna*. Esto producirá un evento de salida con valor $y = \lambda\,(s)$ y un cambio a un nuevo estado

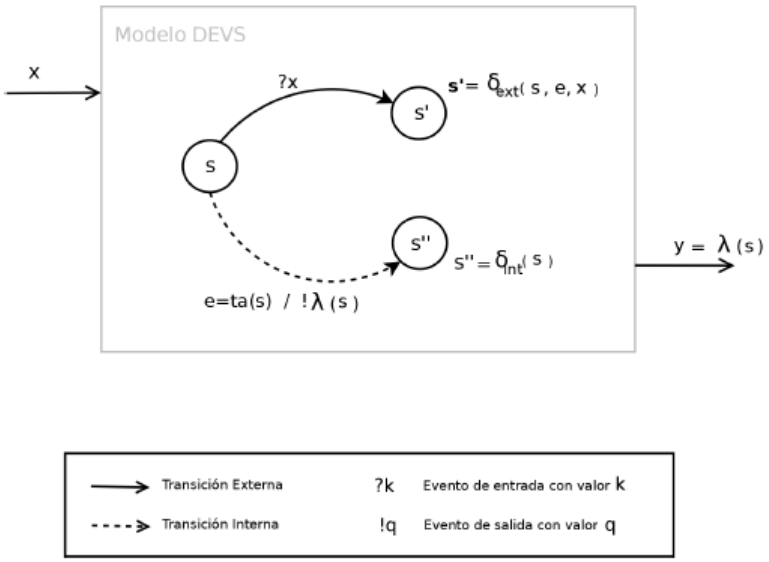

Figure 3.6: Cambio de Estados DEVS

$s'' = \delta_{int}(s)$. Esto es graficado a través de la fecha discontinua que va de s a s''.

En ambos casos, el sistema permanecerá en el nuevo estado, por el tiempo que *ta* lo determine o hasta que otra vez ocurra un evento externo.

Obsérvese en la definición de DEVS presentada, que la función de *avance de tiempo* admite asociar un tiempo 0 o ∞ a un estado $s \in S$. En el primer caso, el sistema sólo podrá permanecer en s, 0 instantes de tiempo. Por tanto, cuando alcance s, inmediatamente sucederá una transición interna ocurriendo un evento de salida y un cambio de estado. Este tipo de estado es llamado *estado transitorio*. De otro modo, cuando el sistema alcance s cuyo tiempo asociado sea ∞, no existirán transiciones internas y permanecerá por siempre en este estado, salvo que, un evento externo ocurra. Este tipo de estado es llamado *estado pasivo*.

De todo esto podemos deducir que las trayectorias de entrada de un DEVS serán segmentos de eventos que ocurrirán en un tiempo cualquiera. Las trayectorias de estados, en tanto, serán seccionalmente constantes, y las trayectorias de salida serán, al igual que las de entradas, segmentos de eventos. Las trayectorias del tiempo transcurrido, en cambio, serán del tipo *diente de sierra*. En la figura 3.7 pueden verse las diferentes trayectorias mencionadas.

Un modelo DEVS acoplado clásico es una estructura:

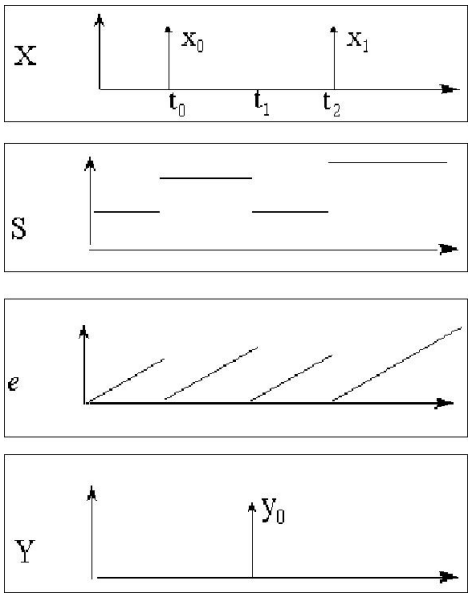

Figure 3.7: Trayectorias DEVS

$$\mathrm{N} = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select \rangle$$

donde:

- X es el conjunto de eventos de entrada
- Y es el conjunto de eventos de salida
- D es el conjunto de referencias a los componentes del modelo acoplado, tal que $\forall d \in D, M_d$ es un modelo DEVS Clásico
- I_d es el conjunto de *componentes influyentes* sobre d tal que,
  $\forall d \in D \cup \{N\}, I_d \subseteq D \cup \{N\} - \{d\}$ y $\forall i \in I_d, Z_{i,d}$ es la *función de traducción* de salidas desde i hacia d con
  $Z_{i,d} : X \rightarrow X_d, sii = N$
  $Z_{i,d} : Y_i \rightarrow Y, sid = N$
  $Z_{i,d} : Y_i \rightarrow X_d, sid \neq N \wedge i \neq N$
- *Select* es una *función de desempate* que arbitra la ocurrencia de eventos simultáneos
  $Select : 2^D \rightarrow D$

Cada modelo M_d recibe eventos de entrada que provienen de otros modelos que también constituyen a N. Por esta causa, estos modelos se donominan *influyentes* sobre d y conforman el conjunto I_d asociado a M_d. Obsérvese que dentro de este

conjunto puede estar incluido el mismo modelo acoplado N pero, M_d **no puede ser influyente de sí mismo**.

N es influyente de M_d si los eventos de entrada en el primero resultan eventos de entrada en el segundo. Por su parte, el conjunto influyente I_N está constituido por los componentes *internamente* influyentes en N. Estos son, aquellos componentes cuyas salidas resultan en eventos de salida del modelo acoplado. En muchos casos, los eventos que puede recibir un componente no son del mismo tipo que aquellos que generan sus influyentes. Para salvar estas diferencias el formalismo cuenta con las *funciones de traducción*.

Puede haber conflictos de simultaneidad. Las transiciones internas en un modelo, generan eventos de salida que a su vez representan eventos de entrada en otros componentes. El conflicto aparece entonces, cuando un componente y alguno de sus influyentes deben llevar a cabo una transición interna. El funcionamiento del sistema cambiará sustancialmente, dependiendo de qué componente ejecute antes esta transición. Por tal motivo, es necesario especificar cuál es el comportamiento adecuado del sistema. La función *Select* viene entonces, a cubrir esta necesidad. Esta función especifica cuál será el componente que tendrá prioridad ante la existencia de más de un componente que deba realizar una transición interna.

3.3.3 Representación en Ecore

La definición del metamodelo DEVS se basa en la especificación descripta en 3.3.2 y se construye con la misma herramienta MediniQVT. La figura 3.8 muestra el diagrama de clases de dicho metamodelo, y las figuras 3.9 y 3.10 su implementeación en formato Ecore.

Brevemente, el cuadro 3.1 describe la relación entre los elementos de la definición formal de un modelo DEVS y los elementos del metamodelo en Ecore.

Definición	**Descripción**	**Ecore (EClass)**
MA	Modelo atómico	AtomicModel
S	Estado	State
X	Evento de entrada	InputEvent
Y	Evento de salida	OutputEvent
δ_{int}	Transición interna	InternalTransition
δ_{ext}	Transición externa	ExternalTransition
λ	Función de salida	OutputFunction
ta	Tiempo de vida	lifeTime(Double)

Table 3.1: Mapeo entre la definición formal de un MA DEVS y un MA DEVS en Ecore.

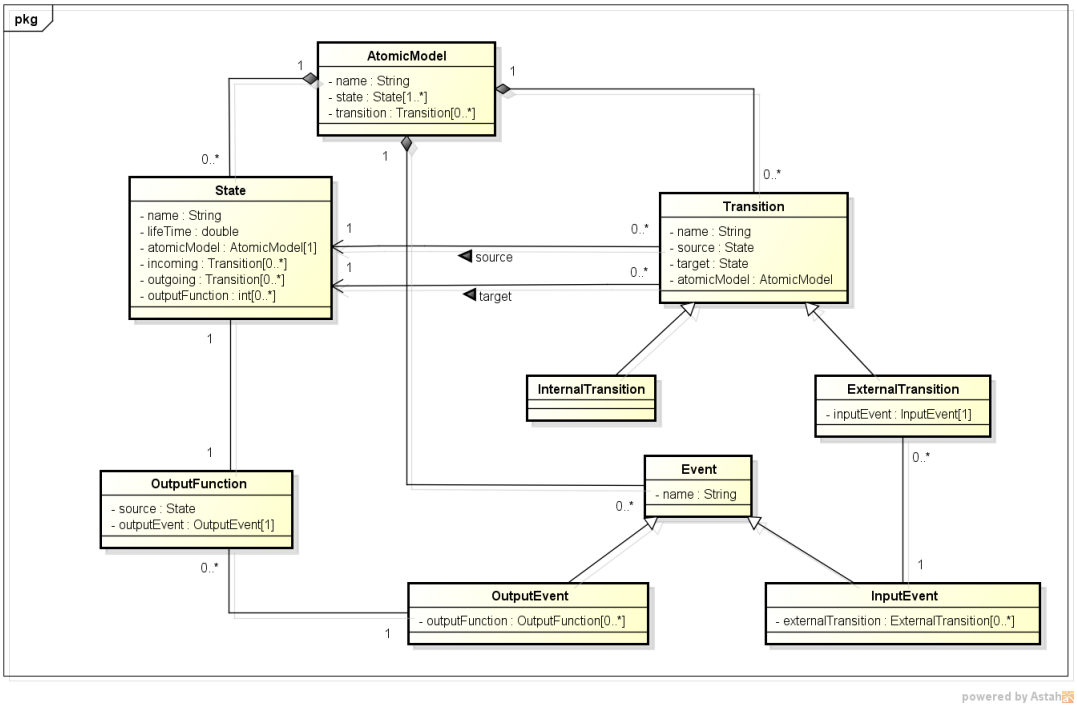

Figure 3.8: Diagrama de clases del metamodelo DEVS.

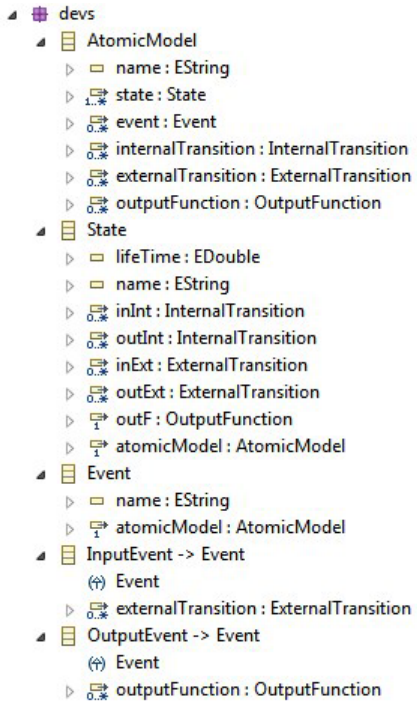

Figure 3.9: Metamodelo DEVS en Ecore (1).

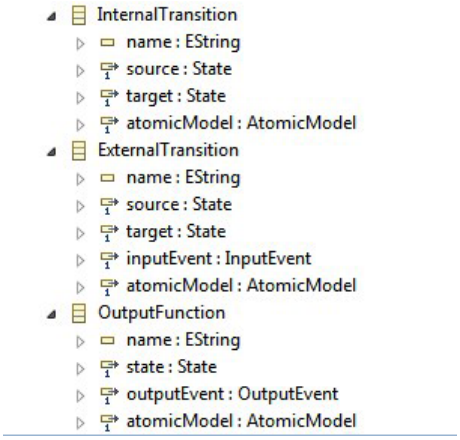

Figure 3.10: Metamodelo DEVS en Ecore (2).

Chapter 4

Aplanado de Máquinas de Estados UML vía Transformación de Modelos

El modelado es una metodología que consiste en reproducir sistemas reales a través de un conjunto de elementos, instrucciones y reglas que representen su estructura y comportamiento. Los elementos estructurales incluyen los componentes del sistema, tales como los puertos de entrada y salida de datos, entidades de estado, conexiones, etc. Mientras que los elementos de comportamiento definen las secuencias de interacciones, reglas de tiempo y operaciones de componentes.

A la hora de modelar un sistema es importante conocer y utilizar todas las utilidades que me provee la herramienta de modelado para poder obtener un modelo correcto y adecuado que permita comprender el sistema, es decir sus elementos y relaciones. Además que permita reducir la complejidad para poder entender sistemas complejos en su totalidad, que facilite la comunicación y le dé claridad. Es por ello que a la hora de modelar una máquina de estados UML es importante que la herramienta permita utilizar estados compuestos para realizar un modelo adecuado del sistema.

Uno de los objetivos principales de este trabajo es proveer un mecanismo que logre aplanar una Máquina de Estados UML, mediante un proceso de transformación de modelos, que transforme una Máquina de Estados UML con estados compuestos (OR - Estados Secuenciales) a una Máquina de Estados UML con estados simples, permitiendo así una posterior transformación a su correspondiente modelo atómico DEVS definida en [1]. Para elaborar dicho proceso de transformación es necesario hacer una revisión de los elementos del formalismo, posteriormente, se define un mapeo entre ellos a través de un conjunto de reglas que denotan como la información de un modelo se traduce a otro. Queda como trabajos futuros considerar el uso de estados ortogonales (AND - Estados Concurrentes). Dicha transformación de modelos es definida con QVT-Relations para luego continuar con las siguientes transformaciones dentro

del contexto de MDA.

Si bien existe una gran variedad de propuestas que especifican como aplanar Máquinas de Estados UML empleando diferentes técnicas, ninguna de ellas esta especificada con lenguajes específicos de transformación de modelos como QVT-Relations que es lo que las diferencia de este trabajo.

4.1 Definición Formal de las ME

Retomando los conceptos de la sección 3.1.1, las ME de UML están basadas en los StateCharts definidos por Harel, compuestos por un conjuntos de estados, transiciones y eventos.

Formalmente, siguiendo [37], la definición de la ME que se utiliza como dominio origen en el proceso se compone de la siguiente manera:

$$\text{ME} = \langle S, \Sigma, T, A \rangle$$

donde:

- S: Es un conjunto de estados no vacíos.
- Σ: Es un conjunto finito de eventos.
- $(A \subseteq \Sigma)$: es el conjunto de acciones, donde $\tau \in A$ representa la acción nula.
- T es un conjunto finito de transiciones que se representan mediante la tupla: $t = (t', s_o, e, c, a, s_d)$, donde
  - t' es el nombre de la transición.
  - $s_o \in S$ es el estado origen.
  - $s_d \in S$ es el estado destino.
  - $e \in \Sigma$ es el evento que dispara la transición.
  - c es una condición de disparo sobre la transición.
  - $a \in A$ es una acción que se ejecuta cuando se efectúa la transición.

Por practicidad, se puede asociar a un estado un tiempo de permanencia. Estos son utilizados para describir un nuevo comportamiento, cuya semántica define el tiempo en que el sistema puede permanecer en un estado. Al transcurrir dicho tiempo debe dispararse una transición definida para tal situación. Este comportamiento también se podría implementar asociando un evento de tiempo a una transición, la cual disparará cuando el estado origen alcanza dicho tiempo. Por un lado, se formaliza la asociación de un tiempo a cada estado, y por otro, se distingue un evento especial $\gamma \in \Sigma$ denominado *evento de tiempo* para ser utilizado en aquellas transiciones que se dispararán al alcanzar el tiempo de permanencia de su estado origen. Denominamos a éstas *transiciones de tiempo*. Sólo una *transición de tiempo* debe definirse por cada estado cuyo

tiempo de permanencia es distinto de ∞.

Se extiende la definición de una ME con los siguientes componentes:

$$\text{ME} = \langle S, \Sigma, T, A, \Pi \rangle$$

donde:

- $\Pi : S \times \Re^{+}_{0,\infty}$ es un conjunto de pares (función) que asocia a cada estado un $\Re^{+}_{0,\infty}$, el cual representa el tiempo de permanencia.
- $\gamma \in \Sigma$, es el *evento de tiempo*,
- Si $\Pi(s) \neq \infty$ entonces debe existir una única *transición de tiempo* con origen s, esto es, $\exists! t\_tiempo = (t', s, \gamma, true, a, s_d) \in T$.

La figura 4.1 muestra un ejemplo de una ME, la cual modela el comportamiento de un sistema en donde las transiciones que se disparan cuando se agota el tiempo de permanencia en su estado origen (*transiciones de tiempo*), se dibujan simplemente asociandole el tiempo de permanencia y la acción de salida, su evento (evento de tiempo *tau*) y condición (*true*) no es necesario especificarlo.

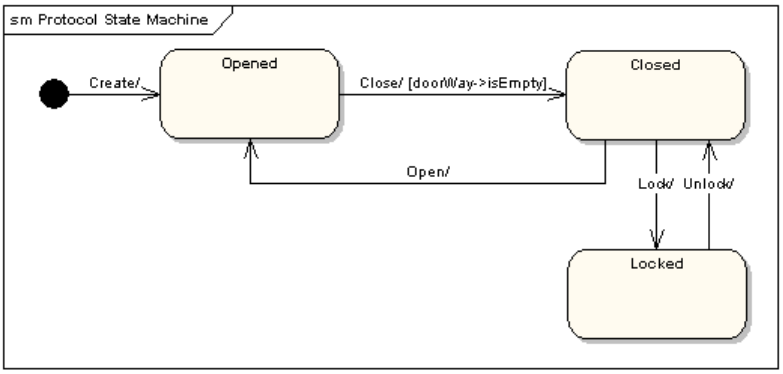

Figure 4.1: Máquina de estados UML

4.2 Implementación de la Transformación con Query/ View/ Transformation

En esta sección se propone una implementación del proceso de transformación de ME UML con estados compuestos a ME UML con estados simples, basada en el lenguaje estándar QVT Relations, previo análisis y comparación de algunos lenguajes de transformación existentes. Más específicamente, se define un conjunto de relaciones QVT

que implementan dicha transformación.

Se utiliza en esta implementación el metamodelo de UML 2.4.1 descripto en el sitio oficial de la OMG. En función de dicha especificación se describen que elementos de UML son utilizados para dar soporte al formalismo de las ME.

Una ejecución de las reglas en QVT toma como parámetros, un modelo (o instancia) de una ME UML que satisface el metamodelo UML en Ecore, y construye un modelo de una ME UML aplanada que satisface el respectivo metamodelo en formato Ecore.

En la sección 4.4 se presenta un ejemplo de aplicación de las reglas.

4.2.1 Metamodelo UML en Ecore

El metamodelo de UML que se utiliza en la transformación está disponible en formato Ecore en el sitio web de la OMG: `http://www.omg.org/spec/MOF/20110701/uml.ecore`. Dicha especificación corresponde a la versión UML 2.4.1.

En función de esta especificación se establecen cuales elementos UML son utilizados para modelar las ME definidas en 4.1, tal como se muestra brevemente en el cuadro 4.1.

Definición	**Descripción**	**Ecore (EClass)**
ME	Máquina de estados	StateMachine
S	Estado	State
Σ	Evento	SignalEvent
A	Acción	FunctionBehavior
T	Transición	Transition
Π	Tiempo de vida	Constraint

Table 4.1: Mapeo entre la definición formal de una ME y los elementos de UML 2.4.1.

4.2.2 Análisis de los Lenguajes de Transformación de Modelos

En esta sección se analizan las propuestas existentes con el fin de fundamentar la elección del lenguaje utilizado en el actual trabajo. El cuadro 4.2 es una muestra de sólo algunos de una diversidad de lenguajes de transformación que se pueden encontrar en la actualidad. Se muestra el gran número de propuestas existentes, que evidencia el gran interés que se ha despertado por este aspecto del paradigma del desarrollo de software dirigido por modelos, así como la enorme actividad de investigación surgida alrededor del mismo.

Existe una variedad muy amplia de lenguajes de transformación, donde cada uno posee características deseables al igual que inconvenientes. Los lenguajes específicos del dominio (Domain Specific Language - DSL), como MT, ofrecen la ventaja de aprovechar todo el potencial del lenguaje *host*, sumándole la expresividad y sofisticación del aparato transformacional. No obstante, esta característica también los hace

Lenguaje/Implementación	**Breve Descripción**
ATL (Atlas Transformation Language)	Lenguaje de transformación de modelos y herramienta para Eclipse desarrollada por el Atlas Group (INRIA).
BOTL(Basic Object-Oriented Transformation Language)	Propuesta gráfica de la Universidad Técnica de München.
ETL (Epsilon Transformacion Language)	Lenguaje definido sobre la plataforma Epsilon, plataforma desarrollada como un conjunto de plug-ins (editores, asistentes, pantallas de configuración, etc.) sobre Eclipse. Dicha plataforma presenta el lenguaje de metamodelado independiente Epsilon Object Language que se basa en OCL, y puede ser utilizado como lenguaje de gestión de modelos o como infraestructura a extender para producir nuevos lenguajes específicos de dominio, como el citado Epsilon Transformation Language (ETL). Además de ETL, Epsilon soporta a Epsilon Comparison Language (ECL) y a Epsilon Merging Language (EML), orientados a la comparación y composición de modelos, respectivamente.
M2T (Model to Text project)	Se focaliza en la generación de artefactos textuales a partir de modelos. Consta de un conjunto de herramientas desarrolladas para la plataforma Eclipse.
MTF (Model Transformation Framework)	Es un conjunto de herramientas desarrollado por IBM que permiten hacer comparaciones, comprobar la consistencia y ejecutar transformaciones entre modelos EMF.
MTL (Model Transformation Language)	Lenguaje de transformación de modelos y herramienta en Eclipse desarrollada por el grupo Triskell.
QVT (Query/View/Transformation)	Especificación estándar de OMG. Está basado en MOF (Meta Object Facility) para lenguajes de transformación en MDA. Implementaciones: QVT−Operational: QVT Operacional de Borland SmartQVT Eclipse M2M QVT-Core: OptimalJ (Descontinuado) QVT-Relations: MediniQVT Eclipse M2M Declarative QVT ModelMorf
RubyTL	Lenguaje de transformación híbrido definido como un lenguaje específico del dominio embebido en el lenguaje de programación Ruby. Diseñado como un lenguaje extensible en el que un mecanismo de plugins permite añadir nuevas características al núcleo de características básicas.

Table 4.2: Lenguajes de transformación de modelos.

poco compatible con otras herramientas que se basen en los estándares de la OMG (QVT, OCL, MOF, etc). Por otra parte, los lenguajes gráficos como UMLX son sumamente intuitivos pero a la vez limitados por la misma conformación de la simbología del lenguaje. Conjuntamente, la descripción de una transformación demasiada compleja puede resultar engorrosa para su utilización. Además, suelen incurrir en costos computacionales más elevados en comparación con lenguajes textuales, debido especialmente a que su representación es más compleja de procesar por su naturaleza gráfica. Los lenguajes de transformación M2T son bastante útiles para realizar implementa-

ciones específicas de plataforma a partir de un modelo independiente de plataforma. Sin embargo, dejan de lado la transformación entre modelos, parte fundamental del desarrollo de software dirigido por modelos. En cualquier caso, actualmente muchas de las propuestas analizadas se encuentran en etapa de desarrollo, lo cual seguramente permitirá observar, en el futuro, mejoras en cada una de ellas.

QVT y los basados en éste, como por ejemplo ATL, tienen la ventaja de ceñirse a un estándar, lo cual los hace mas compatibles con otras tecnologías y comprensibles dada la formalidad de su especificación. Además, múltiples herramientas se han desarrollado que dan soporte a su implementación, entre ellas MediniQVT, la cual es una de las más robustas en la actualidad. Bajo las consideraciones mencionadas y otras comparativas efectuadas el lenguaje seleccionado para el actual trabajo es QVT.

4.2.3 Lenguaje Query/View/Transformation Relations

La OMG ha definido el estándar Query/View/Transformation (QVT) [6] para utilizar en los procesos de desarrollo de software guiados por modelos. QVT consta de tres partes: consulta, vista y transformaciones. El componente de consultas de QVT toma como entrada un modelo y selecciona elementos específicos del mismo. Para la resolución de las consultas y restricciones sobre los modelos, se propone el uso de una versión extendida de OCL 2.0 [40]. La parte de vista es una proyección realizada sobre un modelo y es creada mediante una transformación. Una vista puede verse como el resultado de una consulta sobre un modelo, ofreciendo como resultado un punto de vista de éste, restringiéndolo de acuerdo a alguna condición impuesta. En esta sección se presenta el componente de transformaciones de QVT que tiene como objetivo definir transformaciones. Estas transformaciones describen relaciones entre un metamodelo fuente F y un metamodelo objetivo O. Ambos metamodelos deben estar especificados con el estándar Meta-Object Facility (MOF) [41]. Una vez definida la transformación en QVT, ésta puede ejecutarse para obtener un modelo objetivo, que es una instancia del metamodelo O a partir de un modelo fuente, el cual es una instancia del metamodelo F.

La especificación de QVT 1.0 [6] tiene una naturaleza híbrida declarativa/imperativa, donde la parte declarativa se divide en una arquitectura de dos niveles (ver figura 4.2) constituida por un metamodelo y un lenguaje llamado *Relations* que sigue un paradigma declarativo, el cual soporta concordancia de patrones (Pattern Matching) de objetos complejos y plantillas de creación de objetos, y un metamodelo y un lenguaje llamado *Core*, definido usando mínimas extensiones de MOF y OCL. En este trabajo se hace uso del lenguaje *Relations* que permite especificar relaciones entre metamodelos MOF y de la herramienta para la definición de las transformaciones MediniQVT [42]. Esta herramienta implementa la especificación QVT/Relations en un poderoso motor QVT. Está diseñada para transformaciones de modelos permitiendo un rápido desarrollo, mantenimiento y particularización de reglas de transformación de procesos específicos. Su interfaz está basada en Eclipse y utiliza Eclipse Modeling Framework (EMF) [43] para la representación de los modelos. Además, posee un editor con asistente de código y un depurador de relaciones. MediniQVT exige que

los metamodelos (y modelos) sean escritos en una variante simplificada del estándar MOF, llamada Ecore [44], el cual ha sido definido en el EMF.

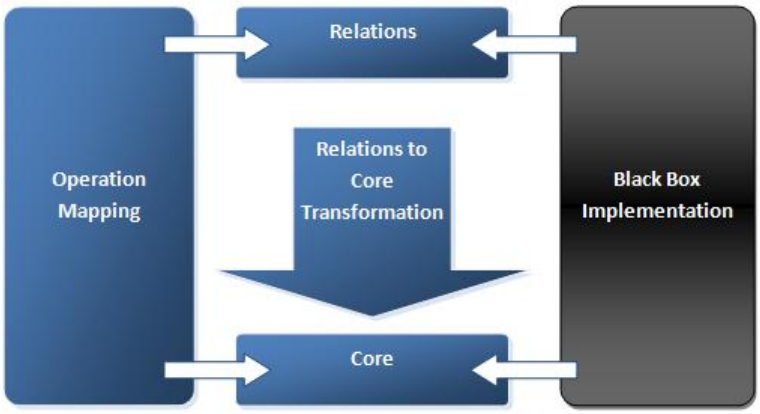

Figure 4.2: Arquitectura QVT.

Características de Eclipse Modeling Framework y Ecore

El lenguaje Ecore es el utilizado por el Eclipse Modeling Framework (EMF). Los metamodelos y modelos usados por EMF se representan con documentos XML. Existen herramientas que tratan automáticamente estos metamodelos y modelos. Por ejemplo, EMF genera código automáticamente para editores de modelos para un lenguaje de modelado, definido con un metamodelo en Ecore. Otro ejemplo es el plug-in de Eclipse para transformar este tipo de modelos, con la definición de transformaciones QVT. Los metamodelos de Ecore son XML, sin embargo EMF posee un editor gráfico de modelos Ecore que facilita su creación.

Las características y elementos del lenguaje de modelado Ecore son los siguientes: el elemento aglutinador (raíz) es el paquete `EPackage`, que contiene físicamente a sus elementos (especificación de `containment`) y, a su vez, éstos pueden estar contenidos en otros paquetes. Hay una fábrica (`EFactory`) por paquete que permite la creación de los elementos del modelo. Las construcciones que describen a un conjunto de elementos (instancias) son clasificadores (`EClassifiers`): `EClass` y `EDataType`.

Ecore especifica las características de las clases (`EClass`), sus características estructurales (`EStructuralFeatures`), atributos (`EAttributes`), operaciones y relaciones (herencia, referencia (`EReference`)). Las `EClasses` tienen superclases y están compuestas por características estructurales (`EStructuralFeatures`: `EReference` y `EAttribute`). Tanto las `EReferences` como los `EAttributes` pueden estar dotados de multiplicidad. Los `EDataTypes` modelan tipos básicos o indivisibles del modelo de datos, y los `EReferences` pueden estar contenidos o ser referencias (punteros). Las

`Operations` modelan operaciones de la interfaz (aunque no se provee implementación para ellas). Todos los elementos heredan de `ENamedElement` (tienen nombre), y de `EModelElement` (elemento del modelo). Además, todo elemento del modelo puede tener asociadas anotaciones (`EAnnotation`): pares nombre/valor para especificaciones extra (por ejemplo, restricciones OCL o cadenas de documentación). La figura 4.3, muestra la especificación del estándar Ecore.

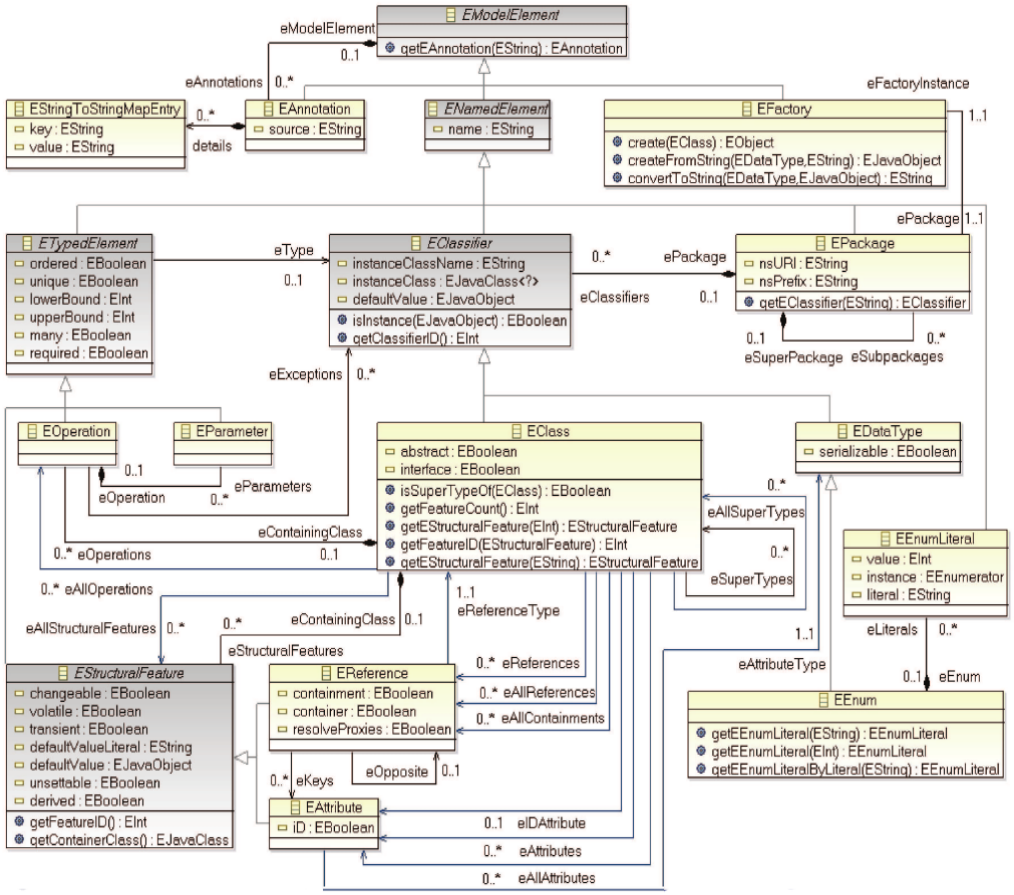

Figure 4.3: Especificación del formato Ecore.

4.3 La Transformación

La definición de las relaciones (reglas de transformación) en QVT-Relations implementan la transformación de una máquina de estados (OR) UML a una máquina de estados plana UML. Dichas reglas de transformación se escriben de forma declarativa y describen relaciones consistentes entre el conjunto de elementos de cada modelo.

Esta consistencia se puede verificar ejecutando la transformación en modo "checkonly" (sólo lectura), retornando "True" si el modelo es consistente según la transformación, o "False" en caso contrario. De la misma manera, se puede ejecutar en modo "enforce" para modificar o construir uno de los modelos, de tal manera que ambos satisfagan las relaciones al final de la ejecución. Las relaciones QVT se definen marcando "checkonly" el modelo UML con estados compuestos, y con "enforce" el modelo UML plano, mapeando así los elementos del modelo origen al modelo destino.

A continuación se muestran las partes más importantes del código de las relaciones definidas para llevar a cabo el proceso de "aplanar" una máquina de estados UML en QVT-Relations. En el anexo A se puede ver la definición completa de las reglas.

1. Mapeo de Regiones.
   La región de la Máquina de estados plana UML se construye según el siguiente mapeo: el primer `MapRegion` es la que se encarga de mapear la región que corresponde a la State Machine manteniendo el mismo nombre. La segunda definición es la que se encarga de mapear las regiones que están en las regiones interiores a la región superior, es decir las regiones que se encuentran dentro de un estado OR. Se puede observar que esta definida de forma recursiva, lo que hace que siempre se mantenga una sola región en el modelo de destino, como si se estuviera subiendo la región hasta el tope de la raíz.

```
/* Map Region*/
top relation MapRegion{
  nregion : String;
  checkonly domain  source s_source :uml::Region {
    name = nregion,
    stateMachine = source_machine:uml::StateMachine{}
  };
  enforce domain target s_target :uml::Region{
    name = nregion,
    stateMachine = target_machine:uml::StateMachine{}
  };
  when{
    MapStatemachine(source_machine,target_machine);
  }
}

/* Map Regions that are in a inner region to the upper region*/
top relation MapRegion2{
  nregion : String;
  checkonly domain  source s_source :uml::Region {
    name = nregion
  };
  enforce domain target s_target :uml::Region{
   name = nregion
  };
  when{
    MapRegion(s_source.state.container,s_target) or MapRegion2(s_source.state.container,s_target);
  }
}
```

2. Mapeo de Estados.
   Los estados de la máquina de estados compuesta UML son mapeados a estados simples al modelo destino contenidos por la misma región, manteniendo el nombre y en el caso que contengan Constraint se los mapea con ella.

```
/* Map State*/
top relation MapState{
  nstate :String;
  checkonly domain  source s_source :uml::State {
    name = nstate,
    container = reg_source:uml::Region{}
  };
  enforce domain target s_target :uml::State{
    name = nstate,
    container = reg_target:uml::Region{}
  };
  when {
    not isFinal(s_source);
    MapRegion(reg_source,reg_target) or MapRegion2(reg_source,reg_target);
    if s_source.stateInvariant.oclIsUndefined() then true else MapConstraintAux(s_source,s_target) endif;
  }
}
```

3. Mapeo de Estados Iniciales.
   Los estados iniciales de la máquina de estados compuesta UML son mapeados de la siguiente manera: En la primer definición se mapea el estado inicial de la State Machine del origen a un estado inicial con el mismo nombre contenido por la misma región en el modelo de destino. En la segunda relación se mapean los estados iniciales contenidos en regiones interiores en el modelo origen, a estados con el mismo nombre contenidos en la misma región. Y se le agrega una Constraint de tiempo 0 que va a ser el tiempo de permanencia en ese estado ya que lo que se busca es crear un estado intermedio que sirva de transición inmediata entre el padre del estado inicial y el estado de destino de la transición de salida del estado inicial en el modelo origen.

```
/* Map PseudoState*/
top relation MapPseudoState{
  npseudostate : String;
  checkonly domain  source s_source :uml::Pseudostate{
    name = npseudostate,
    container = reg_source:uml::Region{}
  };
  enforce domain target s_target :uml::Pseudostate{
    name = npseudostate,
    container = reg_target:uml::Region{}
  };
  when{
    MapRegion(reg_source,reg_target);
  }
}

/* Map PseudoStates in inner regions changing them to states*/
top relation MapPseudoState2{
  npseudostate : String;
  checkonly domain  source s_source :uml::Pseudostate{
    name = npseudostate,
    container = reg_source:uml::Region{}
  };
  enforce domain target s_target :uml::State{
    name = npseudostate,
    container = reg_target:uml::Region{},
    stateInvariant = unv:uml::Constraint{specification = spe:uml::LiteralString{value = '0.0'}}
  };
  when{
    MapRegion2(reg_source,reg_target);
  }
}
```

4. Mapeo de Estados Finales.
   Solo se mapea el Estado Final de la StateMachine manteniendo el mismo nombre y contenido en la misma región. No se mapean los estados finales de los estados compuestos OR ya que, lo que se hace es crear una transición del ultimo estado (estado anterior al FinalState) de un estado compuesto OR al estado destino de la transición de salida del estado compuesto OR.

```
/* Map FinalState*/
top relation MapFinalState{
  nfinalstate : String;
  checkonly domain  source s_source :uml::FinalState{
    name = nfinalstate,
    container = reg_source:uml::Region{}
  };
  enforce domain target s_target :uml::FinalState{
    name = nfinalstate,
    container = reg_target:uml::Region{}
  };
  when{
    MapRegion(reg_source,reg_target);
  }
}
```

5. Mapeo de Transiciones que van de un estado a un estado simple.

```
top relation MapTransition1{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::State{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    not target_s.isComposite();
    MapState(source_s,source_t) and (MapState(target_s,target_t) or MapFinalState(target_s,target_t));
    MapRegion(reg_source,reg_target) or MapRegion2(reg_source,reg_target);
    if s_source.trigger->size()=0 then true else MapTriggerAux(s_source,s_target) endif;
  }
  where{
     if s_source.effect.oclIsUndefined() then true else MapActivityAux(s_source,s_target) endif;
  }
}
```

6. Mapeo de Transiciones que van de un estado a un estado final contenido en un estado OR .

```
top relation MapTransition2{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
```

```
    source = source_s:uml::State{},
    target = target_s:uml::FinalState{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    MapState(target_s.container.state,target_t);
    MapRegion2(reg_source,reg_target);
    MapState(source_s,source_t);
    if s_source.trigger->size()=0 then true else MapTriggerAux(s_source,s_target) endif;
  }
  where{
   if s_source.effect.oclIsUndefined() then true else MapActivityAux(s_source,s_target) endif;
  }
}
```

7. Mapeo de Transiciones que van de un estado a un estado compuesto, las cuales son transformadas a transiciones que van de el estado origen al estado inicial (que es transformado como estado) contenido en el estado compuesto.

```
top relation MapTransition3{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::State{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    target_s.isComposite();
    MapState(source_s,source_t);
    MapPseudoState2(target_s.region->at(1).subvertex
      ->select(v :uml::Vertex| v.incoming->isEmpty())->at(1),target_t);
    MapRegion2(reg_source,reg_target) or MapRegion(reg_source,reg_target);
    if s_source.trigger->size()=0 then true else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then
      true else MapActivityAux(s_source,s_target) endif;
  }
}
```

8. Mapeo de Transiciones que van de un estado inicial a un estado compuesto, las cuales son transformadas a transiciones que van de el estado origen (estado inicial transformado a estado) al estado inicial (que es transformado como estado) contenido en el estado compuesto.

```
top relation MapTransition4{
  ntransition : String;
```

```
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::Pseudostate{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    target_s.isComposite();
    MapPseudoState2(source_s,source_t);
    MapPseudoState2(target_s.region->at(1).subvertex
      ->select(v :uml::Vertex| v.incoming->isEmpty())->at(1),target_t);
    MapRegion2(reg_source,reg_target) or MapRegion(reg_source,reg_target);
    if s_source.trigger->size()=0 then true else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then
     true else MapActivityAux(s_source,s_target) endif;
  }
}
```

9. Mapeo de Transiciones que van de un estado inicial a un estado compuesto, las cuales son transformadas a transiciones que van del estado origen (estado inicial de la máquina de estados que no es transformado como estado) al estado inicial (que es transformado como estado) contenido en el estado compuesto.

```
top relation MapTransition5{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::Pseudostate{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::Pseudostate{},
    target = target_t:uml::State{}
  };
  when{
    target_s.isComposite();
    MapPseudoState(source_s,source_t);
    MapPseudoState2(target_s.region->at(1).subvertex
      ->select(v :uml::Vertex| v.incoming->isEmpty())->at(1),target_t);
    MapRegion2(reg_source,reg_target) or MapRegion(reg_source,reg_target);
    if s_source.trigger->size()=0 then true else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then
     true else MapActivityAux(s_source,s_target) endif;
  }
}
```

10. Mapeo de Transiciones que van de un estado inicial a un estado simple .

```
top relation MapTransition6{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::Pseudostate{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    not target_s.isComposite();
    MapPseudoState2(source_s,source_t) and (MapState(target_s,target_t)
     or MapFinalState(target_s,target_t));
    MapRegion(reg_source,reg_target) or MapRegion2(reg_source,reg_target);
    if s_source.trigger->size()=0 then true else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then
     true else MapActivityAux(s_source,s_target) endif;
  }
}
```

4.4 Ejemplo de Aplicación

En esta sección se describe un ejemplo simplificado del funcionamiento de un Cajero Automático de un Banco (CAB). El comportamiento es especificado mediante una máquina de estados UML (con estados OR), y se muestran en detalle todos los pasos que produce la transformación al MA DEVS correspondiente. Luego, brevemente se muestra la generación de código C++ para la simulación del MA DEVS utilizando la herramienta PowerDevs.

4.4.1 Cajero Automático de un Banco

El ejemplo muestra el funcionamiento simplificado de los cajeros automáticos que se encuentran en las entidades bancarias, siendo éstos máquinas que le permiten a un usuario seleccionar distintos tipos de transacciones bancarias mediante el intercambio de datos a través de una tarjeta magnética. Además, la máquina dispone de mecanismos para diagnosticar fallas y proveer información a un usuario de mantenimiento para llevar a cabo procesos de reparación.

Por razones de espacio, a continuación se describe sólo el comportamiento más relevante del cajero automático. Se considera que el CAB comienza apagado, y una vez que es conectado a la alimentación eléctrica inicia un proceso de prueba para la puesta en marcha el cual dura unos 10 segundos; en caso de falla pasa a fuera de servicio, de lo contrario quedará listo para realizar operaciones bancarias por los usuarios. Cuando el usuario ingresa su tarjeta debe autenticarse, en caso de fallar la autenticación, a los 30 segundos el CAB vuelve a estar listo para operar expulsando la tarjeta ingresada; en caso contrario, presenta al usuario un menú de opciones para comenzar una

transacción. Al finalizar la transacción el usuario decide si volver al menú de opciones o salir del sistema. En caso de fallas, el CAB dispone de un proceso de diagnóstico y reparación que un usuario de mantenimiento se encarga de configurar y ejecutar. Este procedimiento se puede realizar cuando el CAB se encuentra ocioso o fuera de servicio.

4.4.1.1 CAB UML

La figura 4.4 ilustra la máquina de estados del cajero automático utilizando la representación gráfica de UML, para realizar esta representación se utiliza el editor ad-hoc de máquinas de estados que se define en el capitulo 6.

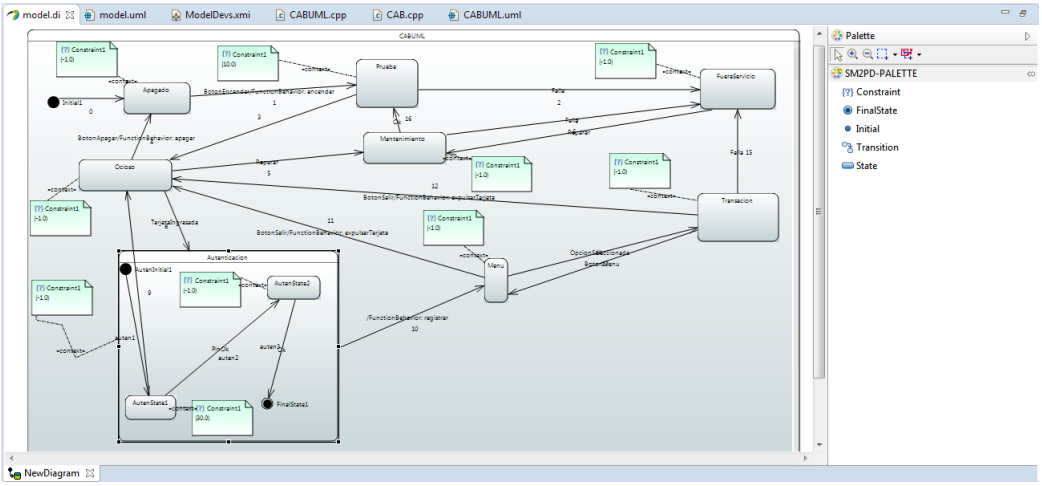

Figure 4.4: Definición de un Cajero Automático de un Banco en UML

A continuación se describe la especificación formal de la ME del CAB.

$$\text{CAB} = \langle S, \Sigma, T, A, \Pi \rangle$$

donde:

- $S : \langle Apagado, Prueba, Fuera\ de\ Servicio, Ocioso, Mantenimiento, Autenticación \langle AutenState1, AutenState2 \rangle, Menú, Transformación \rangle$
- $\Sigma : \langle BotónEncender, BotónApagar, BotónSalir, Ok, BotónMenú, TarjetaIngresada, OpciónSelecc, Falla, Reparar, Mant\text{-}OK, PinOk, Ok \rangle$
- $A : \langle Apagar, Encender, ExpulsarTarjeta, RegistrarUsuario \rangle$
- $T = \langle (Apagado \rightarrow Prueba, Apagado, BotónEncender, true, Encender, Prueba), (Prueba \rightarrow Fuera\ de\ Servicio, Falla, true, \tau, Fuera\ de\ Servicio), (Ocioso \rightarrow Apagado, Ocioso, BotónApagar, true, Apagar, Apagado),$

$(Ocioso \rightarrow Autenticación, Ocioso, TarjetaIngresada, true, \tau, Autenticación)$,
$(AutenState1 \rightarrow Ocioso, Autenticación, \gamma, true, \tau, Ocioso)$,
$(Prueba \rightarrow Ocioso, Prueba, \gamma, true, \tau, Ocioso)$,
$(Ocioso \rightarrow Mantenimiento, Ocioso, Reparar, true, \tau, Mantenimiento)$,
$(Mantenimiento \rightarrow Fuera\ de\ Servicio, Mantenimiento, Falla, true, \tau, Fuera\ de\ Servicio)$,
$(Mantenimiento \rightarrow Prueba, Mantenimiento, Mant\text{-}OK, true, \tau, Prueba)$,
$(Fuera\ de\ Servicio \rightarrow Mantenimiento, Fuera\ de\ Servicio, Reparar, true, \tau, Mantenimiento)$,
$(AutenState1 \rightarrow AutenState2, AutenState1, PinOk, true, \tau, AutenState2)$,
$(Autenticación \rightarrow Menu, Autenticación, Ok, true, RegistrarUsuario, Menú)$,
$(Menú \rightarrow Ocioso, Menú, BotónSalir, true, ExpulsarTarjeta, Ocioso)$,
$(Menú \rightarrow Transacción, Menú, OpciónSeleccionada, true, \tau, Transacción)$,
$(Transacción \rightarrow Menú, Transacción, BotónMenú, true, \tau, Menú)$,
$(Transacción \rightarrow Fuera\ de\ Servicio, Transacción, Falla, true, \tau, Fuera\ de\ Servicio)$,
$(Transacción \rightarrow Ocioso, Transacción, BotónSalir, true, ExpulsarTarjeta, Ocioso)\rangle$

- $\Pi : \langle(Apagado, \infty), (Prueba, 10), (Fuera\ de\ Servicio, \infty), (Ocioso, \infty), (Mantenimiento, \infty), (Autenticación, \infty), (Menú, \infty), (Transacción, \infty), (AutenState1, 30), (AutenState2, \infty)\rangle$

Las figuras 4.5, 4.6, y 4.7 muestran la ME UML del CAB en Ecore, la cual es construida con el editor ad-hoc:

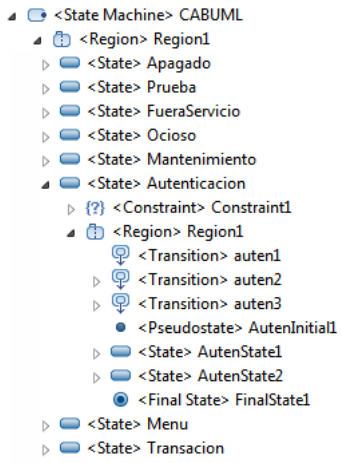

Figure 4.5: Estados del CAB UML en Ecore

4.4.1.2 De CAB UML a CAB DEVS

La ejecución de las reglas de transformación en QVT Relations, toma como argumento el modelo del CAB UML en formato Ecore y crea un modelo atómico DEVS en el mismo formato.

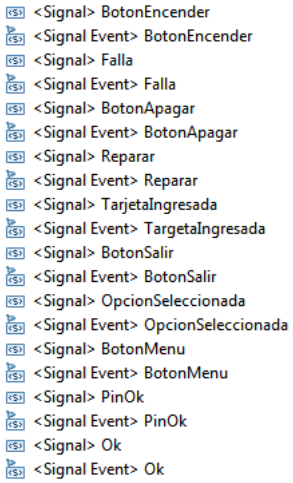

Figure 4.6: Eventos del CAB UML en Ecore

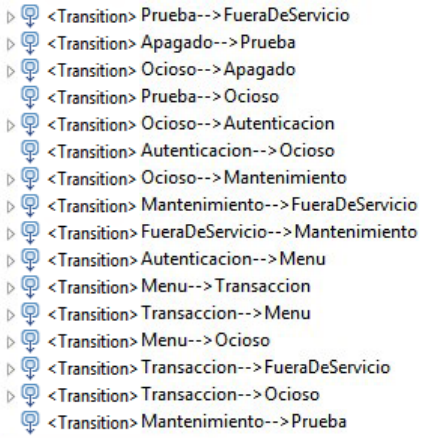

Figure 4.7: Transiciones del CAB UML en Ecore

La figura 4.8 muestra un diagrama del modelo atómico DEVS resultante. Si bien los modelos DEVS no tienen una notación gráfica estandarizada debido a que sus conjuntos pueden ser infinitos, en este caso es posible graficar el modelo dado que la cantidad de elementos transformados son finitos. Por convención, las transiciones internas están dibujadas con líneas punteadas y la función de salida *lambda* es representada implícitamente sobre las transiciones internas especificando la salida en función del estado origen, por ej., $!ExpulsarTarjeta$. Los demás elementos son intuitivos en el diagrama.

Notar, que la transformación de las transiciones que se disparan por la ocurrencia de eventos externos y que tienen acciones como salida, generan un estado intermedio con tiempo de permanencia 0(*cero*) con el fin de producir una salida inmediata de la acción. Por ejemplo, la transición que va desde el estado *Apagado* al estado *Prueba*, produce el estado intermedio $Apagado - Prueba$; cuando éste es alcanzado por la ocurrencia del evento *BotónEncender* se dispara inmediantamente (debido a que el tiempo de vida es cero) una transición interna que cambia el CAB al estado *Prueba*, y además, lanza una salida de la acción *Encender* ($\lambda(Apagado - Prueba) = Encender$).

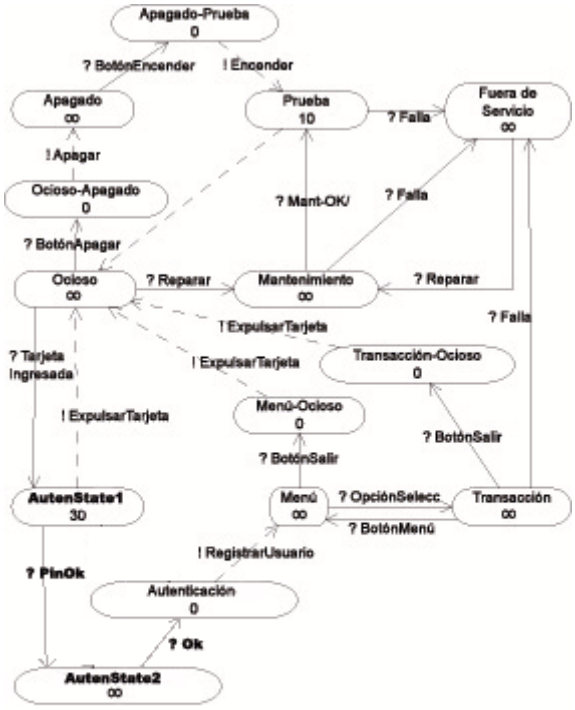

Figure 4.8: Diagrama del Cajero Automático de un Banco en DEVS.

La especificación formal del MA DEVS del CAB obtenido es como sigue:

$$\text{CAB} = \langle X, Y, S, ta, \delta_{ext}, \delta_{int}, \lambda \rangle$$

donde:

- $X = \langle BotónApagar, BotónSalir, Ok, PinOk, BotónMenú, TarjetaIngresada, OpciónSelecc, BotónEncender, Falla, Reparar \rangle$

- $Y = \langle Apagar, ExpulsarTarjeta, RegistrarUsuario, Encender \rangle$
- $S = \langle (d, \sigma) | d \in \langle Apagado, Prueba, Fuera\ de\ Servicio, Ocioso, Mantenimiento, AutenState1, AutenState2, Autenticación,\ Menú, Transacción \rangle, \sigma \in \Re^{+}_{0,\infty} \rangle$
- Transiciones internas:
  $\delta_{int}(Apagado - Prueba, \sigma) = (Prueba, 10)$
  $\delta_{int}(Prueba, \sigma) = (Ocioso, \infty)$
  $\delta_{int}(Ocioso - Apagado, \sigma) = (Apagado, \infty)$
  $\delta_{int}(AutenState1, \sigma) = (Ocioso, \infty)$
  $\delta_{int}(Autenticación, \sigma) = (Menú, \infty)$
  $\delta_{int}(Menú - Ocioso, \sigma) = (Ocioso, \infty)$
  $\delta_{int}(AutenInitial1, \sigma) = (AutenState1, 30)$
  $\delta_{int}(Transacción - Ocioso, \sigma) = (Ocioso, \infty)$
- Transiciones externas:
  $\delta_{ext}(((Apagado, \sigma), t_e), ?BotónEncender) = (Apagado - Prueba, 0)$
  $\delta_{ext}(((Prueba, \sigma), t_e), ?Falla) = (Fuera\ de\ Servicio, \infty)$
  $\delta_{ext}(((Fuera\ de\ Servicio, \sigma), t_e), ?Reparar) = (Mantenimiento, \infty)$
  $\delta_{ext}(((Mantenimiento, \sigma), t_e), ?Falla) = (Fuera\ de\ Servicio, \infty)$
  $\delta_{ext}(((Ocioso, \sigma), t_e), ?Reparar) = (Mantenimiento, \infty)$
  $\delta_{ext}(((Ocioso, \sigma), t_e), ?BotónApagar) = (Ocioso - Apagado, 0)$
  $\delta_{ext}(((Ocioso, \sigma), t_e), ?TarjetaIngresada) = (Autenticación, 30)$
  $\delta_{ext}(((AutenState1, \sigma), t_e), ?PinOk) = (AutenState2, 0)$
  $\delta_{ext}(((AutenState2, \sigma), t_e), ?Ok) = (Autenticación, 0)$
  $\delta_{ext}(((Menó, \sigma), t_e), ?BotónSalir) = (Menú - Ocioso, 0)$
  $\delta_{ext}(((Menú, \sigma), t_e), ?OpciónSelecc) = (Transacción, \infty)$
  $\delta_{ext}(((Transacción, \sigma), t_e), ?BotónMenú) = (Menú, \infty)$
  $\delta_{ext}(((Transacción, \sigma), t_e), ?Falla) = (Fuera\ de\ Servicio, \infty)$
  $\delta_{ext}(((Transacción, \sigma), t_e), ?BotónSalir) = (Transacción - Ocioso, 0)$
- Función de salida:
  $\lambda(Apagado - Prueba, \sigma) = !Encender$
  $\lambda(Ocioso - Apagado, \sigma) = !Apagar$
  $\lambda(Autenticación, \sigma) = !RegistrarUsuario$
  $\lambda(Menú - Ocioso, \sigma) = !ExpulsarTarjeta$
  $\lambda(Transacción - Ocioso, \sigma) = !ExpulsarTarjeta$

Las figuras 4.9, 4.10 y 4.11 ilustran el modelo atómico DEVS en Ecore como resultado de la transformación.

4.4.1.3 CAB en PowerDEVS

El modelo DEVS resultante de la transformación puede ser exportado a distintas herramientas para realizar simulaciones; entre otras, PowerDevs, DEVSJAVA [30], DEVS-C++ [45], DEVSim++ [46], CD++ [47] y JDEVS [48]. Estas herramientas de software ofrecen diferentes características, las cuales incluyen interfaces gráficas y funcionalidades de simulación avanzadas de propósito general y de dominio específico

Figure 4.9: Estados del CAB DEVS en Ecore

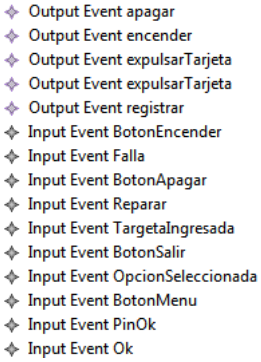

Figure 4.10: Eventos del CAB DEVS en Ecore

para modelos DEVS. En este trabajo se utiliza PowerDevs, por ser una herramienta de código abierto (open source) y sencilla para implementar estos modelos; además, posee versiones tanto para *windows* como para *linux* y es utilizada ampliamente en el ámbito académico para la enseñanza de M&S.

PowerDevs está implementado en C++ (con librerías gráficas QT) y sus modelos DEVS deben ser definidos también en C++. Se compone de varios programas independientes:

- *Model Editor*: contiene la interfaz gráfica que permite, entre otras definiciones de alto nivel, la construcción jerárquica de la estructura de los modelos DEVS (archivos .pdm), los cuales tienen una sintaxis especial.

- *Atomic Editor*: permite definir en C++ (archivos .h y .cpp) el comportamiento de los modelos atómicos DEVS, es decir, las funciones de transición, la función

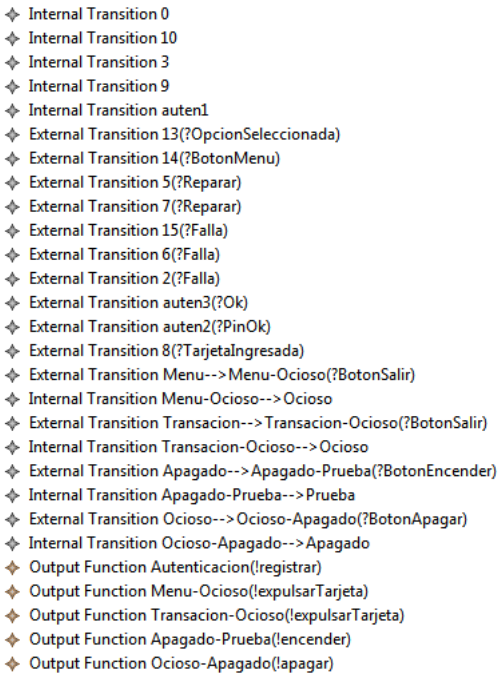

Figure 4.11: Transiciones del CAB DEVS en Ecore

de salida, el tiempo de vida y demás elementos que componen dichos modelos.

- *Preprocessor*: traduce los archivos del *Model Editor* en estructuras con información necesaria para construir el código de simulación, y enlaza los archivos creados con el *Atomic Editor* compilando a un archivo *stand-alone* ejecutable.

- *Simulation Interface*: ejecuta los archivos *stand-alone* ejecutables, permitiendo variar los parámetros de la simulación.

- Una instancia de ejecución de *Scilab* donde los parámetros de la simulación son leídos y los resultados pueden ser exportados.

La exportación de un modelo atómico DEVS en Ecore consiste en producir, por un lado, un archivo (.pdm) que define la estructura del modelo y puede ser abierto por el *Model Editor*; y por otro lado, los archivos que definen el comportamiento (archivos .h y .cpp) de tal manera que puedan ser interpretados por el *Atomic Model*. La transformación de M2T que genera el código esta basado en un template definido en Acceleo.

El proceso de exportación descripto es representado en la figura 4.12.

A continuación se exhibe el código C++ generado correspondiente al archivo de cabecera CAB.h:

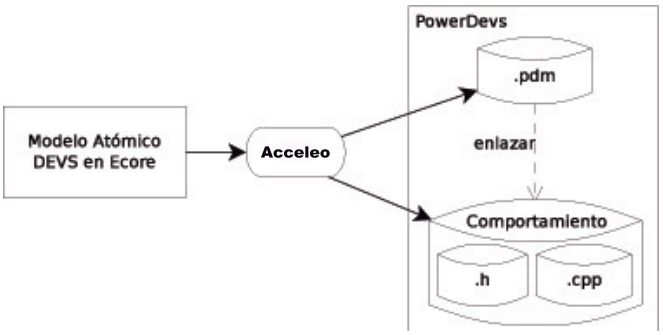

Figure 4.12: Exportación de modelos atómicos DEVS en Ecore a PowerDevs.

```
/* File: CABUML.h */
#if !defined CABUML_h
#define CABUML_h
#include "simulator.h"
#include "event.h"
#include "stdarg.h"

class CABUML: public Simulator {
// Declare the state,
// output variables
// and parameters

char *s;
double sigma;
char *y;
#define INF 1e20

public:
  CABUML(const char *n): Simulator(n) {};
  void init(double, ...);
  double ta(double t);
  void dint(double);
  void dext(Event , double );
  Event lambda(double);
  void exit();
};
#endif
```

El código C++ generado (CAB.cpp) correspondiente al comportamiento del CAB DEVS en Ecore:

```
/* File: CABUML.cpp */
#include "CABUML.h"
void CABUML::init(double t,...) {
  //The 'parameters' variable contains the parameters
  //transferred from the editor.
  va_list parameters;
  va_start(parameters,t);
  //To get a parameter: %Name% = va_arg(parameters,%Type%)
  //where:
  //  %Name% is the parameter name
  //  %Type% is the parameter type
  s = "inicial";
  sigma = INF;
}
```

```
double CABUML::ta(double t) {
  //This function returns a double.
  return sigma;
}

void CABUML::dint(double t) {

  if (strcmp(s,"inicial")== 0) {
    s = "Apagado";
    sigma = INF;
  };

  if (strcmp(s,"Autenticacion")== 0) {
    s = "Menu";
    sigma = INF;
  };

  if (strcmp(s,"Prueba")== 0) {
    s = "Ocioso";
    sigma = INF;
  };

  ...

}

void CABUML::dext(Event x, double t) {
  //The input event is in the 'x' variable.
  //where:
  //     'x.value' is the value (pointer to void)
  //     'x.port' is the port number
  //     'e' is the time elapsed since last transition

  if ( (strcmp(s,"Menu")== 0) &&
   (strcmp((char*)x.value,"OpcionSeleccionada")== 0)) {
    s = "Transacion";
    sigma = INF;
  };

  if ( (strcmp(s,"Transacion")== 0) &&
   (strcmp((char*)x.value,"BotonMenu")== 0)) {
    s = "Menu";
    sigma = INF;
  };

  ...

}

Event CABUML::lambda(double t) {
  //This function returns an Event:
  //     Event(%Value%, %NroPort%)
  //where:
  //     %Value% points to the variable which
  //      contains the value.
  //     %NroPort% is the port number (from 0 to n-1)

  if (strcmp(s,"Autenticacion")== 0) {
    y = "registrar";
    return Event(y,0);
  };

  if (strcmp(s,"Menu-Ocioso")== 0) {
    y = "expulsarTarjeta";
    return Event(y,0);
  };

  ...
```

```
}

void CABUML::exit() {
  //Code executed at the end of the simulation.
}
```

El código correspondiente a la estructura del modelo PowerDEVS (CABUML.pdm):

```
Coupled
  {
  Type = Root
  Name = CABUML
  Ports = 0; 0
  Description =
  Graphic
    {
    Position = 0; 0
    Dimension = 600; 600
    Direction = Right
    Color = 15
    Icon =
    Window = 5000; 5000; 5000; 5000
    }
  Parameters
    {
    }
  System
    {
    Atomic
      {
      Name = CABUML
      Ports = 1 ; 1
      Path = discrete/CABUML.h
      Description = Atomic DEVS model
      Graphic
        {
        Position = -6105 ; -2610
        Dimension = 675 ; 720
        Direction = Right
        Color = 15
        Icon = None
        }
      Parameters
        {
        }
      }
    }
  }
```

Chapter 5

Automatización de las Transformaciones

En este capítulo se va a desarrollar como es todo el proceso de automatización de las transformaciones partiendo de un modelo de máquina de estados UML hasta obtener el código C++ correspondiente al modelo atómico Devs para poder importarlo en la herramienta PowerDevs y realizar una simulación.

Para realizar la automatización se desarrolla un Plugin para Eclipse que va a integrar la implementación de MediniQvt-Relational para eclipse, para poder ejecutar las transformaciones definidas en QVT-Relational. Además integrará la implementación de Acceleo para eclipse, para poder ejecutar las transformaciones de modelos a código C++. El Plugin contará con una interfaz de usuario, para que se pueda dar comienzo a todo el proceso de transformación.

Antes de comenzar a explicar como se realiza el proceso de automatización se estudiarán las herramientas utilizadas para llevar a cabo el proceso.

5.1 Acceleo

Acceleo es un generador de código open-source de la Fundación Eclipse que permite a las personas utilizar un enfoque basado en modelos para la creación de aplicaciones. Es una implementación de MOFM2T basada en el estándar de la OMG, para llevar a cabo la transformación de modelo a texto.

Acceleo es una herramienta de transformación de modelo a texto, con un lenguaje basado en plantillas o templates. Teniendo en cuenta los estándares definidos por EMF, Acceleo brinda un enfoque MDA simple permitiendo la generación de archivos a partir de modelos UML, MOF o EMF.

Acceleo brinda un asistente para la generación automática de las mismas. Al inicio se debe indicar el metamodelo a ser utilizado y posteriormente se debe seleccionar el modelo en sí. Acceleo está basado en los principales estándares MDA, lo que garantiza su compatibilidad e interoperabilidad con otras aplicaciones, como GMF. Es compatible con XMI, fue diseñado para trabajar con cualquier metamodelo y permite extender la funcionalidad ofrecida mediante la importación de librerías de Java, que pueden utilizarse para agregar funcionalidad a las plantillas y la generación de código. Una vez seleccionado el metamodelo y el modelo de entrada, se debe indicar las plantillas utilizadas en el proceso de generación. Luego se guarda la cadena indicando su nombre y la ubicación donde se alojará.

5.2 Que es un Plugin para Eclipse

Los plug-ins de eclipse son módulos que permiten agregar funcionalidad al entorno de desarrollo Eclipse. La plataforma Eclipse permite a los desarrolladores extender la funcionalidad a través de los plug-ins.
El desarrollo de un plugin se basa en la definición de extensiones y puntos de extensión. Cuando un plug-in quiere permitir que otros plug-ins extiendan o personalicen partes de su funcionalidad, se define un punto de extensión. Un punto de extensión declara un contrato, definido en XML y interfaces JAVA, que las extensiones deben cumplir. Los plug-ins que quieran conectarse a ese punto de extensión deben implementar el contrato en su extensión. Algunas extensiones son solo declarativas; es decir, que no agregan código como por ejemplo un punto de extensión que agregue atajos del teclado. Otros puntos de extensión permiten añadir elementos a la interfaz de usuarios para agregar cierta funcionalidad que sera implementada por un controlador asociado al punto de extensión. Los puntos de extensión del plug-in son definidos por defecto en un archivo denominado plugin.xml.

En las siguientes secciones se detallan todas las etapas de automatización del proceso de transformación.

5.3 Implementación Código Java para Ejecutar Transformaciones QVT-Relational

En esta sección se proporciona parte del código más importante para poder ejecutar transformaciones definidas en Medini QVT-Relational. En el anexo B se puede ver la implementación completa.

Las transformaciones definidas en QVT-Relational van de modelos UML a modelos UML (Proceso de aplanado) o de modelos UML a modelos DEVS principalmente. Estos modelos se basan en sus metamodelos UML y DEVS correspondientes como se definió en capítulos anteriores. Para poder ejecutar las transformaciones definidas sobre estos modelos en java, es necesario proporcionar al motor de Medini-QVT la

implementación de las clases de los metamodelos.

El código java de las clases correspondientes a los metamodelos ecore se obtiene utilizando el generador de código que nos proporciona Eclipse Modeling Framework (EMF).

A continuación se muestra parte del código generado de algunos de los elementos del metamodelo DEVS definido en la sección 3.3.3.

1. Interface State.java.

```
package devs;

import org.eclipse.emf.common.util.EList;

import org.eclipse.emf.ecore.EObject;

 /**
 * <!-- begin-user-doc -->
 * A representation of the model object '<em><b>State</b></em>'.
 * <!-- end-user-doc -->
 *
 * <p>
 * The following features are supported:
 * <ul>
 *   <li>{@link devs.State#getLifeTime <em>Life Time</em>}</li>
 *   <li>{@link devs.State#getName <em>Name</em>}</li>
 *   <li>{@link devs.State#getIn <em>In</em>}</li>
 *   <li>{@link devs.State#getOut <em>Out</em>}</li>
 *   <li>{@link devs.State#getOutF <em>Out F</em>}</li>
 *   <li>{@link devs.State#getAtomicModel <em>Atomic Model</em>}</li>
 * </ul>
 * </p>
 *
 * @see devs.DevsPackage#getState()
 * @model
 * @generated
 */
 public interface State extends EObject {

   ...

   /**
   * Returns the value of the '<em><b>Name</b></em>' attribute.
   * <!-- begin-user-doc -->
   * <p>
   * If the meaning of the '<em>Name</em>' attribute isn't clear,
   * there really should be more of a description here...
   * </p>
   * <!-- end-user-doc -->
   * @return the value of the '<em>Name</em>' attribute.
   * @see #setName(String)
   * @see devs.DevsPackage#getState_Name()
   * @model
   * @generated
   */
   String getName();

   /**
   * Sets the value of the '{@link devs.State#getName <em>Name</em>}' attribute.
   * <!-- begin-user-doc -->
   * <!-- end-user-doc -->
   * @param value the new value of the '<em>Name</em>' attribute.
   * @see #getName()
```

```
   * @generated
   */
   void setName(String value);

   ...

 } // State
```

2. Implementación StateImpl.java.

```
package devs.impl;

/**
* <!-- begin-user-doc -->
* An implementation of the model object '<em><b>State</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link devs.impl.StateImpl#getLifeTime <em>Life Time</em>}</li>
*   <li>{@link devs.impl.StateImpl#getName <em>Name</em>}</li>
*   <li>{@link devs.impl.StateImpl#getIn <em>In</em>}</li>
*   <li>{@link devs.impl.StateImpl#getOut <em>Out</em>}</li>
*   <li>{@link devs.impl.StateImpl#getOutF <em>Out F</em>}</li>
*   <li>{@link devs.impl.StateImpl#getAtomicModel <em>Atomic Model</em>}</li>
* </ul>
* </p>
*
* @generated
*/
public class StateImpl extends MinimalEObjectImpl.Container implements State {

  ...

  /**
  * The default value of the '{@link #getName() <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
  * @ordered
  */
  protected static final String NAME_EDEFAULT = null;

  /**
  * The cached value of the '{@link #getName() <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
  * @ordered
  */
  protected String name = NAME_EDEFAULT;

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  public String getName() {
    return name;
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
```

```
        * @generated
        */
        public void setName(String newName) {
          String oldName = name;
          name = newName;
          if (eNotificationRequired())
            eNotify(new ENotificationImpl(this, Notification.SET, DevsPackage.STATE__NAME, oldName, name));
        }

        ...

      } //StateImpl
```

Una vez implementados los metamodelos, se define una clase en java que es la encargada de inicializar el motor de Medini-QVT pasandole los metamodelos, modelos y transformaciones definidas para que las ejecute. Parte del código se define a continuación:

```
/*
* Copyright (c) 2007 ikv++ technologies ag
* All rights reserved.
*/
package mediniTransformation;

...

import de.ikv.medini.qvt.QVTProcessorConsts;
import devs.DevsPackage;

/**
* Class, which demonstrated how to use medini QVT inside your Java application. See the TODOs in
* the source code for assistance.
*
* @author Hajo Eichler <eichler@ikv.de>
*/
public class UsingMediniQVTfromApplications {

  private ILog logger;

  private EMFQvtProcessorImpl processorImpl;

  protected ResourceSet resourceSet;

  ...

  /**
  * Example usage of the class {@link UsingMediniQVTfromApplications}.
  *
  * @param args
  *            are ignored here!
  */
  public static void main(String[] args) {

    /*
    // initialize the engine
    UsingMediniQVTfromApplications simpleQVT = new UsingMediniQVTfromApplications(System.out);

    simpleQVT.launch();
    */
  }

  public void launch(String source, String target,String script,String traces,
   String ndirection, String ntransformation, String nameFile) {

    // initialize resource set of models
    this.resourceSet = new ResourceSetImpl();
```

```
    // TODO: collect all necessary packages from the metamodel(s)
    Collection<EPackage> metaPackages = new ArrayList<EPackage>();
    this.collectMetaModels(metaPackages);

    // make these packages known to the QVT engine
    this.init(metaPackages);

    // load the example model files - TODO: adjust the filename!
    Resource resource1 = this.getResource(source);
    Resource resource2 = this.getResource(target);


    // Collect the models, which should participate in the transformation.
    // You can provide a list of models for each direction.
    // The models must be added in the same order as defined in your transformation!
    Collection<Collection<Resource>> modelResources = new ArrayList<Collection<Resource>>();
        Collection<Resource> firstSetOfModels = new ArrayList<Resource>();
    Collection<Resource> secondSetOfModels = new ArrayList<Resource>();
    modelResources.add(firstSetOfModels);
    modelResources.add(secondSetOfModels);
    firstSetOfModels.add(resource1);
    secondSetOfModels.add(resource2);

    // tell the QVT engine, which transformation to execute - there might be more than one in
    // the QVT file!
    String transformation = ntransformation; // TODO: adjust
    // give the direction of the transformation (according to the transformation definition)
    String direction = ndirection; // TODO: adjust

    // just do it ;-)
    try {
      this.transform(qvtRuleSet, transformation, direction);
    } catch (Throwable throwable) {
      throwable.printStackTrace();
    }

  }

  /**
  * Add all metamodel packages of models/qvt script
  *
  * @param metaPackages
  * @return
  */
  protected void collectMetaModels(Collection<EPackage> metaPackages) {
    metaPackages.add(UMLPackage.eINSTANCE);
    metaPackages.add(DevsPackage.eINSTANCE);
  }

  ...

}
```

5.4 Implementación Código Java para Ejecutar Transformaciones M2T-Acceleo

En esta sección se proporciona parte del código más importante para poder ejecutar transformaciones (M2T) definidas en Acceleo. En el anexo C se puede ver la implementación completa.

A continuación se muestra parte del código para inicializar el motor de Acceleo y poder iniciar la generación de código C++ (código fuente de programa en PowerDevs)

a partir del modelo DEVS obtenido.

```
/*******************************************************************************
* Copyright (c) 2008, 2012 Obeo.
* All rights reserved. This program and the accompanying materials
* are made available under the terms of the Eclipse Public License v1.0
* which accompanies this distribution, and is available at
* http://www.eclipse.org/legal/epl-v10.html
*
* Contributors:
*     Obeo - initial API and implementation
*******************************************************************************/
package M2T.main;

...

import org.eclipse.acceleo.engine.event.IAcceleoTextGenerationListener;
import org.eclipse.acceleo.engine.generation.strategy.IAcceleoGenerationStrategy;
import org.eclipse.acceleo.engine.service.AbstractAcceleoGenerator;

/**
* Entry point of the 'Generate' generation module.
*
* @generated
*/
public class Generate extends AbstractAcceleoGenerator {
  /**
  * The name of the module.
  *
  * @generated
  */
  public static final String MODULE_FILE_NAME = "/M2T/main/generate";

  /**
  *  The name of the templates that are to be generated.
  *
  * @generated
  */
  public static final String[] TEMPLATE_NAMES = { "generate_h", "generate_pdm", "generate_cpp" };

  /**
  * This allows clients to instantiates a generator with all required information.
  *
  * @param model
  *            We'll iterate over the content of this element to find Objects matching the first parameter
  *            of the template we need to call.
  * @param targetFolder
  *            This will be used as the output folder for this generation : it will be the base path
  *            against which all file block URLs will be resolved.
  * @param arguments
  *            If the template which will be called requires more than one argument taken from the model,
  *            pass them here.
  * @throws IOException
  *             This can be thrown in two scenarios : the module cannot be found, or it cannot be loaded.
  * @generated
  */
  public Generate(EObject model, File targetFolder,
  List<? extends Object> arguments) throws IOException {
    initialize(model, targetFolder, arguments);
  }

  /**
  * This can be used to launch the generation from a standalone application.
  *
  * @param args
  *            Arguments of the generation.
  * @generated
  */
```

```
  public static void main(String[] args) {
    try {
      if (args.length < 2) {
      System.out.println("Arguments not valid : {model, folder}.");
    } else {
      URI modelURI = URI.createFileURI(args[0]);
      File folder = new File(args[1]);

      List<String> arguments = new ArrayList<String>();

      /*
      * If you want to change the content of this method, do NOT forget to change the "@generated"
      * tag in the Javadoc of this method to "@generated NOT". Without this new tag, any compilation
      * of the Acceleo module with the main template that has caused the creation of this class will
      * revert your modifications.
      */

      /*
      * Add in this list all the arguments used by the starting point of the generation
      * If your main template is called on an element of your model and a String, you can
      * add in "arguments" this "String" attribute.
      */

      Generate generator = new Generate(modelURI, folder, arguments);

      /*
      * Add the properties from the launch arguments.
      * If you want to programmatically add new properties, add them in "propertiesFiles"
      * You can add the absolute path of a properties files, or even a project relative path.
      * If you want to add another "protocol" for your properties files, please override
      * "getPropertiesLoaderService(AcceleoService)" in order to return a new property loader.
      * The behavior of the properties loader service is explained in the Acceleo documentation
      * (Help -> Help Contents).
      */

      for (int i = 2; i < args.length; i++) {
        generator.addPropertiesFile(args[i]);
      }

      generator.doGenerate(new BasicMonitor());
    }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  ...

}
```

5.5 Implementación de interfaz de usuario del Plugin Eclipse - Inicio del Proceso de Transformación

La implementación de la interfaz de usuario del Plugin se dividirá en dos partes:

En una primera etapa se define un Popup que desplegará una opción para iniciar el proceso de transformación sobre un modelo UML a un modelo DEVS con su correspondiente controlador que iniciará la transformación cuando el usuario seleccione la opción en el Popup.
Parte del código se muestra a continuación:

1. Implementación del Popup en plugin.xml.

```
<extension point="org.eclipse.ui.commands">
  <category name="NSD Category"  id="com.nsd.NavigatorPopup.commands.category"/>
  <command name="SM2D RUN" categoryId="com.nsd.NavigatorPopup.commands.category"
   id="com.nsd.NavigatorPopup.commands.navigatorPopupCommand"/>
</extension>
<extension point="org.eclipse.ui.handlers">
  <handler commandId="com.nsd.NavigatorPopup.commands.navigatorPopupCommand"
   class="com.nsd.navigatorpopup.NavigatorPopupHandler"/>
</extension>
<extension point="org.eclipse.ui.menus">
  <menuContribution locationURI="popup:org.eclipse.jdt.ui.PackageExplorer?after=additions">
    <menu label="SM2D Transformation"  icon="icons/SM2D-16x16.png">
      <visibleWhen>
        <with variable="activeMenuSelection">
          <iterate ifEmpty="false">
            <adapt type="org.eclipse.core.resources.IResource">
              <test property="org.eclipse.core.resources.name" value="*.uml" />
            </adapt>
          </iterate>
        </with>
      </visibleWhen>
      <command commandId="com.nsd.NavigatorPopup.commands.navigatorPopupCommand"
       id="com.nsd.NavigatorPopup.menus.navigatorPopupCommand">
      </command>
    </menu>
  </menuContribution>
  <menuContribution locationURI="popup:org.eclipse.ui.navigator.ProjectExplorer#PopupMenu?after=additions">
    <menu label="SM2D Transformation">
      <command commandId="com.nsd.NavigatorPopup.commands.navigatorPopupCommand"
       id="com.nsd.NavigatorPopup.menus.navigatorPopupCommand">
      </command>
    </menu>
  </menuContribution>
</extension>
```

   En la figura 5.1 se puede observar la interfaz gráfica del Popup que permite iniciar el proceso de transformación. Al hacer click en SM2D RUN se obtiene el modelo DEVS obtenido en la transformación.

2. Implementación del Controlador.
   El controlador se encarga de inicializar el motor de medini e iniciar la transformación del modelo UML para obtener el modelo DEVS.

```
// initialize the engine medini
UsingMediniQVTfromApplications simpleQVT = new UsingMediniQVTfromApplications(System.out);
String script = FileUtils.umlCompositeToUmlSimple;
String transformation = "CSCtoSSC";
String direction = "target";
simpleQVT.launch(source,target,script,traces,direction,transformation,fileName);
script = FileUtils.umlToDevs;
transformation = "UmlToDevs";
direction = "smDevs";
simpleQVT.launch(target,target2,script,traces,direction,transformation,fileName);
```

En la segunda parte se define otro Popup que desplegará la opción de ejecutar la transformación del modelo DEVS obtenido por la ejecución de la transformación de la primer parte a código C++, con el controlador asociado que iniciará la transformación. Parte del código se muestra a continuación:

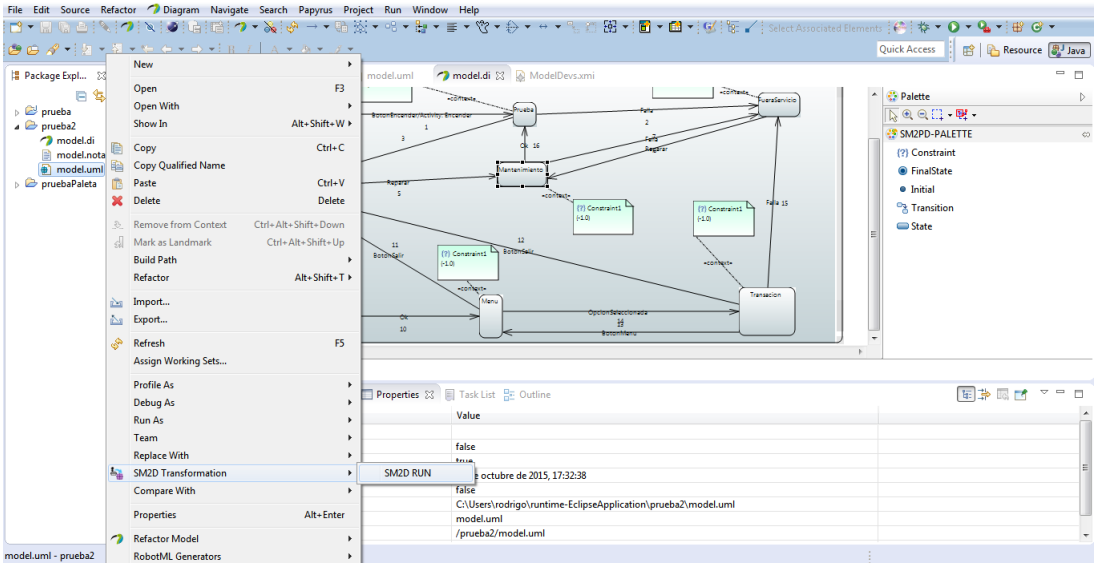

Figure 5.1: Interfaz gráfica del Popup para iniciar la transformación de MEUML a DEVS.

1. Implementación del Popup en plugin.xml.

```
<extension point="org.eclipse.ui.commands">
  <category name="NSD Category"  id="com.nsd.NavigatorPopup2.commands.category"/>
  <command name="D2PD Run Transformation" categoryId="com.nsd.NavigatorPopup2.commands.category"
   id="com.nsd.NavigatorPopup2.commands.navigatorPopupCommand"/>
</extension>
<extension point="org.eclipse.ui.handlers">
  <handler commandId="com.nsd.NavigatorPopup2.commands.navigatorPopupCommand"
   class="com.nsd.navigatorpopup2.NavigatorPopupHandler"/>
</extension>
<extension point="org.eclipse.ui.menus">
  <menuContribution locationURI="popup:org.eclipse.jdt.ui.PackageExplorer?after=additions">
    <menu label="D2PD Transformation"  icon="icons/D2PD-16x16.png">
      <visibleWhen>
        <with variable="activeMenuSelection">
          <iterate ifEmpty="false">
            <adapt type="org.eclipse.core.resources.IResource">
              <test property="org.eclipse.core.resources.name" value="*.xmi" />
            </adapt>
          </iterate>
        </with>
      </visibleWhen>
      <command commandId="com.nsd.NavigatorPopup2.commands.navigatorPopupCommand"
       id="com.nsd.NavigatorPopup2.menus.navigatorPopupCommand">
      </command>
    </menu>
  </menuContribution>
  <menuContribution locationURI="popup:org.eclipse.ui.navigator.ProjectExplorer#PopupMenu?after=additions">
    <menu label="D2PD Transformation">
      <command commandId="com.nsd.NavigatorPopup2.commands.navigatorPopupCommand"
       id="com.nsd.NavigatorPopup2.menus.navigatorPopupCommand">
```

```
      </command>
    </menu>
  </menuContribution>
</extension>
```

En la figura 5.2 se puede observar la interfaz gráfica del Popup que permite iniciar el proceso de transformación. Al hacer click en D2PD Run Transformation se obtiene el código fuente de la simulación para poder ejecutarla en PowerDEVS.

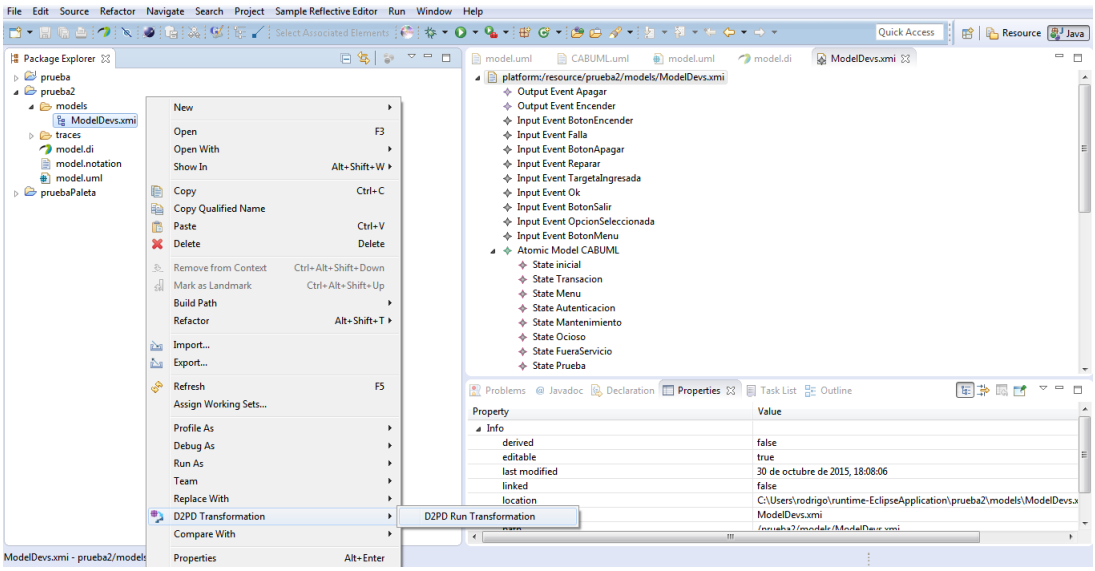

Figure 5.2: Interfaz gráfica del Popup para iniciar la transformación de DEVS a código PowerDEVS.

2. Implementación del Controlador.
   El controlador se encarga de inicializar el motor de acceleo e iniciar la transformación del modelo DEVS para obtener el código C++ para importar en la herramienta PowerDevs.

```
//launch acceleo for transformation M2T
  URI modelURI = URI.createFileURI(source);
  File folder = new File(target);

  List<String> arguments = new ArrayList<String>();

  Generate generator = new Generate(modelURI, folder, arguments);

  generator.doGenerate(new BasicMonitor());
```

Chapter 6

Editor de Máquinas de Estados UML - Plugin para Eclipse

Hasta ahora construimos una herramienta capaz de transformar modelos de máquinas de estados UML simples o compuestas a modelos de simulación Devs, y a partir de allí obtener el código fuente del programa para ejecutar una simulación en la herramienta PowerDevs.

En este capitulo se desarrolla una nueva funcionalidad que permitirá a la herramienta abarcar y dar soporte absolutamente a todo el proceso de diseño, transformación y ejecución de modelos de ME UML.

En particular se desarrolla un Plugin para eclipse que implementa un editor adhoc que permita diseñar ME UML y obtener un modelo de ME UML basado en el metamodelo UML y a partir de allí iniciar todo el proceso de transformación. Es muy importante dar soporte a un editor específico para la herramienta, ya que permite visualizar y entender fácilmente el problema que se está modelando, además acompaña al poder gráfico de UML ya que no es lo mismo que escribir el modelo directamente en formato XMI basado en el metamodelo. Además, el editor limita al usuario a utilizar solo los elementos UML que forman parte de la actual propuesta para luego dar inicio a la transformación, es decir, solo los elementos pertenecientes a máquinas de estados UML con estados OR, lo que reducirá el error a la hora de diseñar por parte del usuario.

El editor es implementado gracias a la integración con el Plugin Papyrus, que permite implementar tus propias paletas con los elementos necesarios basados en un metamodelo, en este caso basado en el metamodelo de ME UML y luego importarlas a la herramienta para poder construir modelos.

Papyrus es un herramienta Open Source basado en eclipse bajo la licencia EPL. Es

una herramienta de edición gráfica para modelos UML2 definido por la OMG, cuyo objetivo es implementar al 100% la especificación de la OMG. Puede ser utilizado como una herramienta independiente o como un plugin para eclipse. Papyrus ofrece editores de diagramas de lenguajes de modelado basados en EME entre ellos UML2 y SysML, permite a los usuarios extenderlo definiendo sus propios editores.

6.1 Implementación del editor de ME UML

En esta sección se proporciona parte del código más importante para poder integrar papyrus a la herramienta y implementar un editor específico de máquinas de estados UML.

Una paleta en papyrus es un conjunto de elementos UML disponibles para construir un diagrama en particular. Se encuentra en la parte derecha del editor de diagramas de papyrus como se muestra en la figura 6.1.

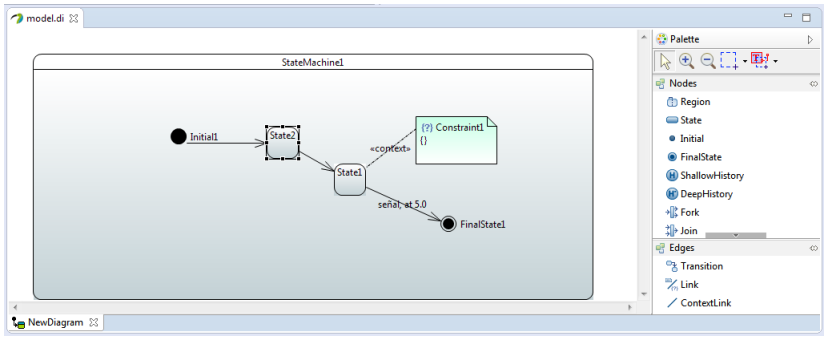

Figure 6.1: Paleta general de una SC en Papyrus.

La implementación del editor se divide en dos partes: La primer parte implementa la paleta en un documento XML en donde se define cada uno de los elementos para la misma, como también los íconos de cada elementos, etc. Papyrus incluye un editor de paletas con interfaz gráfica que genera el archivo XML. Mientras que la segunda parte define un punto de extensión para integrar la paleta definida al plugin y asociarla a los diagramas de máquinas de estados UML.

A continuación se muestra parte del código que implementa el editor de máquinas de estados UML.

1. Definición de la Paleta.xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<paletteDefinition>
  <content>
```

```
    <drawer iconpath="platform:/plugin/org.eclipse.papyrus.uml.diagram.common/icons/local_desc_edit.gif"
     id="drawer_1438295447119" name="SM2PD-PALETTE">
      <aspectTool description="Create Constraint"
       iconpath="platform:/plugin/org.eclipse.uml2.uml.edit/icons/full/obj16/Constraint.gif"
       id="createConstraintCreationTool_1438295727721"
        name="Constraint" refToolId="createConstraintCreationTool"/>
      <aspectTool description="FinalState"
       iconpath="platform:/plugin/org.eclipse.uml2.uml.edit/icons/full/obj16/FinalState.gif"
       id="createFinalStateCreationTool_1438295733317"
        name="FinalState" refToolId="createFinalStateCreationTool"/>
      <aspectTool description="Initial"
       iconpath="platform:/plugin/org.eclipse.uml2.uml.edit/icons/full/obj16/Pseudostate_initial.gif"
       id="createInitialCreationTool_1438295735688"
        name="Initial" refToolId="createInitialCreationTool"/>
      <aspectTool description="Transition"
       iconpath="platform:/plugin/org.eclipse.uml2.uml.edit/icons/full/obj16/Transition_local.gif"
       id="createTransitionCreationTool_1438295797242"
        name="Transition" refToolId="createTransitionCreationTool"/>
      <aspectTool description="State"
       iconpath="platform:/plugin/org.eclipse.uml2.uml.edit/icons/full/obj16/State.gif"
       id="createStateCreationTool_1438295867851"
        name="State" refToolId="createStateCreationTool"/>
    </drawer>
  </content>
</paletteDefinition>
```

2. Definición del Punto de Extensión.

```
<extension
point="org.eclipse.papyrus.uml.diagram.common.paletteDefinition">
  <paletteDefinition ID="SM2DPlugin.paletteDefinition1"
   class="org.eclipse.papyrus.uml.diagram.common.service.PluginPaletteProvider"
   icon="platform:/plugin/org.eclipse.gmf.runtime.diagram.ui/icons/group.gif"
   name="SM2DPlugin.paletteDefinition1" path="palettes/tesis.Palette.xml">
    <Priority name="Highest"></Priority>
    <editor id="org.eclipse.papyrus.uml.diagram.statemachine "></editor>
  </paletteDefinition>
</extension>
```

El editor *ah doc* de SC correspondiente a la implementación del plugin, se puede observar en la figura 6.2.

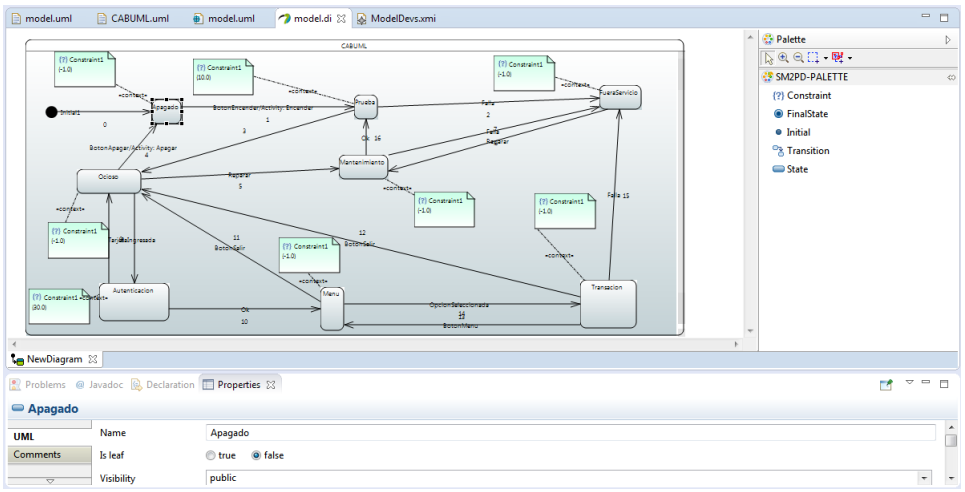

Figure 6.2: Editor ad-hoc de MEUML.

Chapter 7

Pruebas

En este capítulo se describen los métodos y tipos de pruebas utilizados en el desarrollo del Editor de máquinas de Estados UML.

7.1 Pruebas del Editor de Máquinas de Estados UML

El Editor de máquinas de estados UML esta basado en la integración de Papyrus como se explica en unidades anteriores y en la limitación del mismo en cuanto a cantidad de elementos del diagrama de SC que se pueden utilizar. Esto hace que la integridad de los modelos creados con el editor desarrollado esta garantizada por el plugin Papyrus que se basa en la especificación de los metamodelos UML.
Este correcto comportamiento, se puede probar utilizando el método de prueba de Caja Negra o también conocido como método de Entrada/Salida, en donde se proponen datos de entrada (en este caso se crean elementos en el editor gráfico) y se observa si la salida (documento XML que representa el modelo modelado) cumple con la especificación del metamodelo UML.

Se puede observar a continuación que la salida del editor cumple con la especificación de UML definida en el capítulo 4 con respecto a la entrada.

1. Entrada de la Prueba.

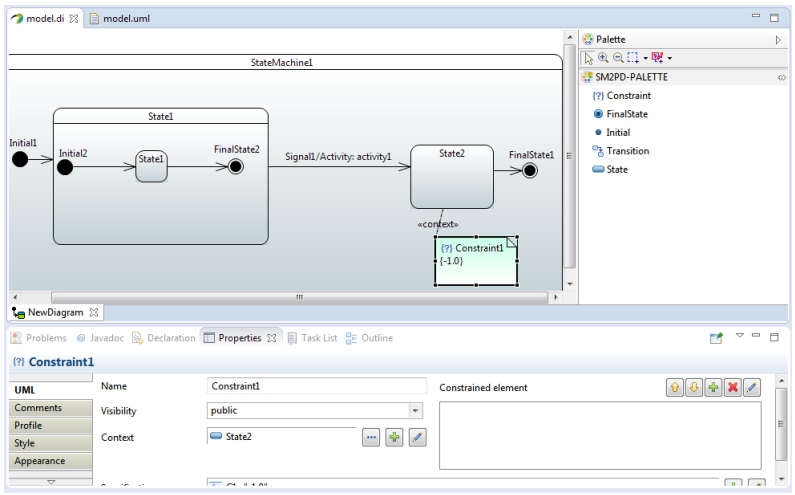

Figure 7.1: Prueba Editor MEUML.

2. Salida de la Prueba.

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:uml="http://www.eclipse.org/uml2/2.1.0/UML" xmi:id="_1UDlYXEGEeSZBOnXec2vrw">
  <packagedElement xmi:type="uml:StateMachine" xmi:id="_tkFAQDV6EeWglZZTpfwP1Q"
  name="StateMachine1" isReentrant="false">
    <region xmi:id="_to_VcDV6EeWglZZTpfwP1Q" name="Region1">
      <transition xmi:id="_HPl7IHDiEeWj2r9gciUOcA" name=""/>
      <transition xmi:id="_IFscsHDiEeWj2r9gciUOcA" name=""/>
      <transition xmi:id="_JHvQcIJlEeW-2uhjXSLWSg"
      source="_BcKEOIJlEeW-2uhjXSLWSg" target="_CFWLwIJlEeW-2uhjXSLWSg"/>
      <transition xmi:id="_J3cPEIJlEeW-2uhjXSLWSg"
      source="_CFWLwIJlEeW-2uhjXSLWSg" target="_C-y8QIJlEeW-2uhjXSLWSg">
        <effect xmi:type="uml:Activity" xmi:id="_gCP9AIJoEeW-2uhjXSLWSg"
        name="activity1" classifierBehavior="_PNtMUIJpEeW-2uhjXSLWSg">
          <ownedBehavior xmi:type="uml:FunctionBehavior" xmi:id="_PNtMUIJpEeW-2uhjXSLWSg"
          name="FunctionBehavior1"/>
        </effect>
        <trigger xmi:id="_9BpWcIJmEeW-2uhjXSLWSg" name="Trigger1" event="_8re5MIJmEeW-2uhjXSLWSg"/>
      </transition>
      <transition xmi:id="_K4kU4IJlEeW-2uhjXSLWSg"
      source="_C-y8QIJlEeW-2uhjXSLWSg" target="_AluzIIJlEeW-2uhjXSLWSg"/>
      <subvertex xmi:type="uml:FinalState" xmi:id="_AluzIIJlEeW-2uhjXSLWSg" name="FinalState1"/>
      <subvertex xmi:type="uml:Pseudostate" xmi:id="_BcKEOIJlEeW-2uhjXSLWSg" name="Initial1"/>
      <subvertex xmi:type="uml:State" xmi:id="_CFWLwIJlEeW-2uhjXSLWSg" name="State1">
        <region xmi:id="_CRIvkIJmEeW-2uhjXSLWSg" name="Region2">
          <transition xmi:id="_RD_f4IJmEeW-2uhjXSLWSg"
          source="_CRWyAIJmEeW-2uhjXSLWSg" target="_DJxAsIJmEeW-2uhjXSLWSg"/>
          <transition xmi:id="_R8ywIIJmEeW-2uhjXSLWSg"
          source="_DJxAsIJmEeW-2uhjXSLWSg" target="_Dxw0OIJmEeW-2uhjXSLWSg"/>
          <subvertex xmi:type="uml:Pseudostate" xmi:id="_CRWyAIJmEeW-2uhjXSLWSg" name="Initial2"/>
          <subvertex xmi:type="uml:State" xmi:id="_DJxAsIJmEeW-2uhjXSLWSg" name="State1"/>
          <subvertex xmi:type="uml:FinalState" xmi:id="_Dxw0OIJmEeW-2uhjXSLWSg" name="FinalState2"/>
        </region>
      </subvertex>
      <subvertex xmi:type="uml:State" xmi:id="_C-y8QIJlEeW-2uhjXSLWSg" name="State2">
        <stateInvariant xmi:id="_GMF3UIJoEeW-2uhjXSLWSg" name="Constraint1">
          <specification xmi:type="uml:LiteralString" xmi:id="_GMHFcIJoEeW-2uhjXSLWSg"
```

```
          name="C1" value="-1.0"/>
        </stateInvariant>
      </subvertex>
    </region>
  </packagedElement>
  <packagedElement xmi:type="uml:SignalEvent" xmi:id="_8re5MIJmEeW-2uhjXSLWSg"
  name="SignalEvent1" signal="_8b8BUIJmEeW-2uhjXSLWSg"/>
  <packagedElement xmi:type="uml:Signal" xmi:id="_8b8BUIJmEeW-2uhjXSLWSg" name="Signal1"/>
</uml:Model>
```

Chapter 8

Conclusiones y Trabajos Futuros

Conclusiones

Las SCs de UML constituyen un mecanismo para especificar el comportamiento de sistemas mediante una representación gráfica. Estos diagramas son compactos, expresivos y proveen la capacidad de modelar no sólo sistemas simples sino también complejos sistemas reactivos. Son muchas las herramientas desarrolladas que dan soporte a las últimas versiones de UML, las cuales proveen abundantes prestaciones gráficas que facilitan el modelado de los sistemas. Sin embargo, carece de la capacidad de ejecución y simulación de los modelos dinámicos, lo cual limita el análisis del comportamiento del sistema en un escenario real.

Por otro lado, DEVS esta fundado sobre los principios de teoría de sistemas y ha procurado su desarrollo mediante la ingeniería basada en componentes desde sus inicios. Con los avances de UML en los últimos años, la comunidad de DEVS ha dedicado esfuerzos para definir un mapeo entre los elementos de DEVS y los de UML. Es prominente la conclusión de que DEVS es mas riguroso y expresivo que UML, pero debido a la manipulación de conjuntos posiblemente infinitos que puede contener un modelo, carece de una notación gráfica, la cual es muy requerida por profesionales en la industria. La posibilidad de integrar en una misma herramienta estos dos formalismos con un editor de máquinas de estados ad-hoc propio, permitiendo modelar SC UML con estados OR para poder aprovechar la potencia gráfica de UML y el poder de simulación de las herramientas DEVS dio lugar a la actual propuesta.

Algunas de las propuestas han intentado integrar DEVS y UML, pero ninguna presenta una solución formal y una implementación basada en la definición de metamodelos haciendo uso de estándares. En este trabajo se presenta un mecanismo que permite integrar y automatizar todo el proceso de transformación entre los dos formalismos con distintas herramientas y tecnologías, distintas bases teóricas, pero unidos por un mismo propósito que es el dar soluciones a problemas reales a través de la

creación y análisis de modelos abstractos, abarcando las etapas de diseño, transformación y ejecución. Para el cual se implementa un editor gráfico de SC UML, además se define un proceso de transformación de modelos que mapea elementos de SC UML con estados OR a elementos de SC UML simple (plana) que luego son mapeados a elementos de modelos DEVS, explotando las cualidades gráficas de uno y la especificidad y rigurosidad del otro, resultando en un modelo de simulación que puede ser ejecutado y analizado por un gran número de herramientas específicas. El resultado de la transformación es un modelo DEVS en formato Ecore que satisface su correspondiente metamodelo. Este modelo provee un conjunto de información suficiente para poder ser interpretado y ejecutado por distintas herramientas y motores de simulación DEVS ya existentes, como PowerDEVS, DEVSJAVA, DEVS/C++, o DEVS.net.

Trabajos Futuros

Como trabajos futuros se prevén las siguientes extensiones y mejoras:

- Permitir modelar máquinas de estados UML compuestas con estados AND. Es decir estados compuestos concurrentes, permitiendo una mayor jerarquización del modelo.
- Permitir al plugin ejecutar simulaciones obtenidas con el motor PowerDevs. De esta forma todo el proceso te automatización se complementaria con la etapa de simulación.
- Validar las transformaciones empíricamente o en un contexto formal mediante técnicas de simulación y bisimulación.
- Enriquecer el editor ad-hoc con elementos subyacentes de las herramientas de simulación de sistemas discretos como por ejemplo: generadores de distribuciones de probabilidad, recolectores de estadísticas, etc.

Bibliography

[1] A. Gonzalez, C. D. Luna, R. Cuello, M. Perez, and M. Daniele, "Metamodel-based transformation from UML state machines to DEVS models," in *XL Latin American Computing Conference, CLEI 2014, Montevideo, Uruguay, September 15-19, 2014*. IEEE, 2014, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1109/CLEI.2014.6965145

[2] H. G. Molter, "Discrete event system specification," in *SynDEVS Co-Design Flow*. Springer Fachmedien Wiesbaden, 2012, pp. 9–42. [Online]. Available: http://dx.doi.org/10.1007/978-3-658-00397-5_2

[3] S. J. Mellor, A. N. Clark, and T. Futagami, "Guest editors' introduction: Model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 14–18, 2003.

[4] *Object Management Group*, Object Management Group Std., Last access: May 2015. [Online]. Available: http://www.omg.org

[5] J. Miller and J. Mukerji, "Mda guide version 1.0.1," Object Management Group (OMG), Tech. Rep., 2003.

[6] OMG, *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1*, Object Management Group Std., Rev. 1.1, 2011. [Online]. Available: http://www.omg.org/spec/QVT/1.1/

[7] *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1*, Object Management Group Std., Rev. 2.4.1, 2011. [Online]. Available: http://www.omg.org/spec/UML/2.4.1

[8] B. P. Zeigler and S. Vahie, "Devs formalism and methodology: Unity of conception/diversity of application," in *Proceedings of the 25th Conference on Winter Simulation*, ser. WSC '93. New York, NY, USA: ACM, 1993, pp. 573–579. [Online]. Available: http://doi.acm.org/10.1145/256563.256724

[9] B. Zeigler, H. Praehofer, and T. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000. [Online]. Available: http://books.google.com.uy/books?id=REzmYOQmHuQC

[10] E. Kofman and S. Junco, "Quantized State Systems. A DEVS Approach for Continuous System Simulation," *Transactions of SCS*, vol. 18, no. 3, pp. 123–132, 2001. [Online]. Available: files/qss.pdf

[11] S. Bendre and H. S. Sarjoughian, "Discrete-event behavioral modeling in sesm: Software design and implementation," in *Advanced Simulation Technology Conference*, 2005, pp. 23–28.

[12] P. P. Vladimir and P. Slavícek, "Towards devs meta language," in *Industrial Simulation Conference*, 2006, pp. 69–73.

[13] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, "Devsml: automating devs execution over soa towards transparent simulators." in *SpringSim (2)*, M. J. Ades, Ed. SCS/ACM, 2007, pp. 287–295. [Online]. Available: http://dblp.uni-trier.de/db/conf/springsim/springsim2007-2.html#MittalRZ07

[14] *JavaML*. [Online]. Available: http://www.badros.com/greg/JavaML

[15] J. de Lara and H. Vangheluwe, "Using atom3 as a meta-case tool." in *ICEIS*, 2002, pp. 642–649. [Online]. Available: http://dblp.uni-trier.de/db/conf/iceis/iceis2002.html#LaraV02

[16] K. Choi, S. Jung, H. Kim, D.-H. Bae, and D. Lee, "Uml-based modeling and simulation method for mission-critical real-time embedded system development," in *IASTED Conf. on Software Engineering*, P. Kokol, Ed. IASTED/ACTA Press, 2006, pp. 160–165.

[17] S. Y. Hong and T. G. Kim, "Embedding uml subset into object-oriented devs modeling process," in *Society of Modeling and Computer Simulation International*, 2004, pp. 161 – 166.

[18] D. Zinoviev, "Mapping DEVS models onto UML models," *CoRR*, vol. abs/cs/0508128, 2005. [Online]. Available: http://arxiv.org/abs/cs/0508128

[19] D. Huang and H. S. Sarjoughian, "Software and simulation modeling for real-time software-intensive systems," in *DS-RT*. IEEE Computer Society, 2004, pp. 196–203.

[20] S. Borland, *Transforming Statechart Models to DEVS*. McGill University, 2004. [Online]. Available: http://books.google.com.ar/books?id=M2v7SgAACAAJ

[21] F. Châne, N. Giambiasi, and J.-L. Paillet, "From devs model to timed automata." in *Software Engineering Research and Practice*, H. R. Arabnia and H. Reza, Eds. CSREA Press, 2004, pp. 498–504. [Online]. Available: http://dblp.uni-trier.de/db/conf/serp/serp2004-2.html#ChaneGP04

[22] S. Schulz, T. Ewing, and J. Rozenblit, "Discrete event system specification (devs) and statemate statecharts equivalence for embedded systems modeling," in *ECBS*. IEEE Computer Society, 2000, p. 308.

[23] D. Harel and M. Politi, *Modeling Reactive Systems with Statecharts: The Statemate Approach*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1998.

[24] A. Tolk and J. A. Muguira, "M&s within the model driven architecture," in *Interservice/Industry Training, Simulation, and Education Conference*, 2004.

[25] J. L. Risco-Martín, J. M. de la Cruz, S. Mittal, and B. P. Zeigler, "eudevs: Executable uml with devs theory of modeling and simulation." *Simulation*, vol. 85, no. 11-12, pp. 750–777, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/simulation/simulation85.html#Risco-MartinCMZ09

[26] S. Mittal, "Extending DoDAF to allow integrated DEVS-based modeling and simulation," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 3, no. 2, pp. 95–123, Apr. 2006. [Online]. Available: http://dx.doi.org/10.1177/875647930600300204

[27] S. Mittal, B. Zeigler, and M. Hwang, *W3C XML Schema for Finite Deterministic (FD) DEVS Models*, Std., Last access: May 2015. [Online]. Available: http://www.saurabh-mittal.com/fddevs/

[28] J. Barnett, R. Akolkar, R. Auburn, M. Bodell, D. C. Burnett, J. Carter, S. McGlashan, T. Lager, M. Helbing, R. Hosn, T. Raman, K. Reifenrath, N. Rosenthal, and J. Roxendal, *State Chart XML (SCXML): State Machine Notation for Control Abstraction - W3C Candidate Recommendation 13 March 2014*, World Wide Web Consortium (W3C) Std., Mar. 2014. [Online]. Available: http://www.w3.org/TR/scxml/

[29] S. Mittal, "Xfd-devs: A demostration." [Online]. Available: http://www.saurabh-mittal.com/fddevs/xfddevs.avi

[30] B. P. Zeigler and H. Sarjoughian, "Introduction to devs modeling and simulation with java: A simplified approach to hla-compliant distributed simulations," *Arizona Center for Integrative Modeling and Simulation*, 2000.

[31] S. Mittal, "Devs unified process for integrated development and testing of service oriented architectures," Ph.D. dissertation, Tucson, AZ, USA, 2007, aAI3255023.

[32] E. Ho, "Creating a text-basededitor for eclipse," 2003. [Online]. Available: http://www.ccs.neu.edu/home/lieber/courses/cs4500/f05/project/pluginDevExample.pdf

[33] M. Balsiger, "A quick-start tutorial to eclipse plug-in development," 2010. [Online]. Available: http://scg.unibe.ch/archive/projects/Bals10b-EclipsePlugins.pdf

[34] A. Wasowski, "Flattening statecharts without explosions," *SIGPLAN Not.*, vol. 39, no. 7, pp. 257–266, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/998300.997200

[35] A. Gonzalez, M. Uva, and M. Frutos, "Refactoring java code by transformation rules in qvt-relation," in *2013 XXXIX Latin American Computing Conference (CLEI), Caracas (Naiguata), Venezuela, October 7-11, 2013*, 2013, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1109/CLEI.2013.6670658

[36] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987.

[37] M. von der Beeck, "A structured operational semantics for uml-statecharts," *Software and Systems Modeling*, vol. V1, no. 2, pp. 130–141, December 2002. [Online]. Available: http://dx.doi.org/10.1007/s10270-002-0012-8

[38] Y. Jin, R. Esser, and J. W. Janneck, "A method for describing the syntax and semantics of uml statecharts." *Software and System Modeling*, vol. 3, no. 2, pp. 150–163, 2004. [Online]. Available: http://dblp.uni-trier.de/db/journals/sosym/sosym3.html#JinEJ04

[39] D. Latella, I. Majzik, and M. Massink, "Towards a formal operational semantics of uml statechart diagrams," in *Proceedings of the IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*. Deventer, The Netherlands, The Netherlands: Kluwer, B.V., 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=646816.708760

[40] OMG, *Object Constraint Language Specification, version 2.0*, Object Modeling Group, June 2005. [Online]. Available: http://fparreiras/papers/OCLSpec.pdf

[41] *OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1*, Object Management Group Std., Rev. 2.4.1, 2011. [Online]. Available: http://www.omg.org/spec/MOF/2.4.1

[42] *Medini QVT*, ikv++ Technologies, Last access: May 2015. [Online]. Available: http://www.ikv.de/

[43] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[44] *Ecore metamodel specification*, Object Management Group, Last access: May 2015. [Online]. Available: http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html

[45] H. Cho and Y. Cho, *DEVSC++ Reference Guide*, The University of Arizona, 1997.

[46] T. G. Kim, *DEVSim++ Users Manual. C++ Based Simulation with Hierarchical Modular DEVS Models*, Korea Advance Institute of Science and Technology, 1994.

[47] G. Wainer, "Cd++: A toolkit to develop devs models," *Softw. Pract. Exper.*, vol. 32, no. 13, pp. 1261–1306, 2002. [Online]. Available: http://dx.doi.org/10.1002/spe.482

[48] J.-B. Filippi and P. Bisgambiglia, "Jdevs: an implementation of a devs based formal framework for environmental modelling," in *Environmental Modelling and Software*, 2004, pp. 261–274.

[49] M. Kay, *SAXON. The XSLT and XQuery Processor*, Saxonica, Last access: May 2015. [Online]. Available: http://saxon.sourceforge.net

Apéndices

Appendix A

Especificación de la Transformación en Query/View/Transformation Relations

Código completo de las relaciones definidas para llevar a cabo el proceso de "aplanar" una máquina de estados UML en QVT-Relations.

```
--transform composite state char to simple state char
transformation CSCtoSSC (source:uml, target:uml){

  /* Map State Machine */
  top relation MapStatemachine{
    nmachine : String;
    checkonly domain  source s_source :uml::StateMachine {
      name = nmachine
    };
    enforce domain target s_target :uml::StateMachine {
      name = nmachine
    };
  }

  /* Map Region*/
  top relation MapRegion{
    nregion : String;
    checkonly domain  source s_source :uml::Region {
      name = nregion,
      stateMachine = source_machine:uml::StateMachine{}
    };
```

```
  enforce domain target s_target :uml::Region{
    name = nregion,
    stateMachine = target_machine:uml::StateMachine{}
  };
  when{
    MapStatemachine(source_machine,target_machine);
  }
}

/* Map Regions that are in a inner region to the upper region*/
top relation MapRegion2{
  nregion : String;
  checkonly domain  source s_source :uml::Region {
    name = nregion
  };
  enforce domain target s_target :uml::Region{
    name = nregion
  };
  when{
    MapRegion(s_source.state.container,s_target) or
      MapRegion2(s_source.state.container,s_target);
  }
}

/* Map State*/
top relation MapState{
  nstate :String;
  checkonly domain  source s_source :uml::State {
    name = nstate,
    container = reg_source:uml::Region{}
  };
  enforce domain target s_target :uml::State{
    name = nstate,
    container = reg_target:uml::Region{}
  };
  when {
    not isFinal(s_source);
    MapRegion(reg_source,reg_target) or
      MapRegion2(reg_source,reg_target);
    if s_source.stateInvariant.oclIsUndefined() then true
      else MapConstraintAux(s_source,s_target) endif;
  }
}

/* Helps map constraints*/
top relation MapConstraintAux{
```

```
  checkonly domain  source s_source :uml::State{
    stateInvariant = v_s:uml::Constraint{}
  };
  enforce domain target s_target :uml::State{
    stateInvariant = v_t:uml::Constraint{}
  };
  when{
    MapConstraint(v_s,v_t);
  }
}

/* Map Constraint*/
top relation MapConstraint{
  nsi : String;
  checkonly domain  source s_source :uml::Constraint{
    name = nsi,
    specification = v_s:uml::LiteralString{}
  };
  enforce domain target s_target :uml::Constraint{
    name = nsi,
    specification = v_t:uml::LiteralString{}
  };
  when{
    MapSpecification(v_s,v_t);
  }
}

/* Map Specification*/
top relation MapSpecification{
  nsi : String;
  checkonly domain  source s_source :uml::LiteralString{
    name = nsi,
    value = val:String{}
  };
  enforce domain target s_target :uml::LiteralString{
    name = nsi,
    value = val:String{}
  };
}

/* Map ActivityAux*/
relation MapActivityAux{
  nactivity : String;
  checkonly domain  source s_source :uml::Transition{
    name = nactivity,
    effect = efs:uml::Activity{}
```

```
  };
  enforce domain target s_target :uml::Transition{
    name = nactivity,
    effect = eft:uml::Activity{}
  };
  where{
    MapActivity(efs,eft);
  }
}

/* Map Activity*/
relation MapActivity{
  nactivity : String;
  checkonly domain  source s_source :uml::Activity{
    name = nactivity,
    classifierBehavior = cb_source:uml::FunctionBehavior{}
  };
  enforce domain target s_target :uml::Activity{
    name = nactivity,
    classifierBehavior = cb_target:uml::FunctionBehavior{}
  };
  where{
    MapFunctionBehavior(cb_source,cb_target);
  }
}

/* Map Behavior*/
relation MapFunctionBehavior{
  nbehavior : String;
  checkonly domain  source s_source :uml::FunctionBehavior{
    name = nbehavior
  };
  enforce domain target s_target :uml::FunctionBehavior{
    name = nbehavior
  };
}

/* Map TriggerAux*/
top relation MapTriggerAux{
  ntrigger : String;
  checkonly domain  source s_source :uml::Transition{
    name = ntrigger,
    trigger = ts:uml::Trigger{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntrigger,
```

```
    trigger = tt:uml::Trigger{}
  };
  when{
    MapTrigger(ts,tt);
  }
}

/* Map Trigger*/
top relation MapTrigger{
  ntrigger : String;
  checkonly domain  source s_source :uml::Trigger{
    name = ntrigger,
    event = e_source:uml::SignalEvent{}
  };
  enforce domain target s_target :uml::Trigger{
    name = ntrigger,
    event = e_target:uml::SignalEvent{}
  };
  when{
    MapSignalEvent(e_source,e_target);
  }
}

/* Map Event*/
top relation MapSignalEvent{
  nevent : String;
  checkonly domain  source s_source :uml::SignalEvent{
    name = nevent,
    signal = signal_source:uml::Signal{}
  };
  enforce domain target s_target :uml::SignalEvent{
    name = nevent,
    signal = signal_target:uml::Signal{}
  };
  when{
    MapSignal(signal_source,signal_target);
  }
}

/* Map Signal*/
top relation MapSignal{
  nsignal : String;
  checkonly domain  source s_source :uml::Signal{
    name = nsignal
  };
  enforce domain target s_target :uml::Signal{
```

```
    name = nsignal
  };
}

/* Map FinalState*/
top relation MapFinalState{
  nfinalstate : String;
  checkonly domain  source s_source :uml::FinalState{
    name = nfinalstate,
    container = reg_source:uml::Region{}
  };
  enforce domain target s_target :uml::FinalState{
    name = nfinalstate,
    container = reg_target:uml::Region{}
  };
  when{
    MapRegion(reg_source,reg_target);
  }
}

/* Map PseudoState*/
top relation MapPseudoState{
  npseudostate : String;
  checkonly domain  source s_source :uml::Pseudostate{
    name = npseudostate,
    container = reg_source:uml::Region{}
  };
  enforce domain target s_target :uml::Pseudostate{
    name = npseudostate,
    container = reg_target:uml::Region{}
  };
  when{
    MapRegion(reg_source,reg_target);
  }
}

/* Map PseudoStates in inner regions changing them to
  states*/
top relation MapPseudoState2{
  npseudostate : String;
  checkonly domain  source s_source :uml::Pseudostate{
    name = npseudostate,
    container = reg_source:uml::Region{}
  };
  enforce domain target s_target :uml::State{
    name = npseudostate,
```

```
    container = reg_target:uml::Region{},
    stateInvariant = unv:uml::Constraint{
      specification = spe:uml::LiteralString{value = '0.0'}
    }
  };
  when{
    MapRegion2(reg_source,reg_target);
  }
}

/* Map transition1: map transitions from state to simple
  state*/
top relation MapTransition1{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::State{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    not target_s.isComposite();
    MapState(source_s,source_t) and (MapState(target_s,target_t) or
      MapFinalState(target_s,target_t));
    MapRegion(reg_source,reg_target) or
      MapRegion2(reg_source,reg_target);
    if s_source.trigger->size()=0 then true
      else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then true
      else MapActivityAux(s_source,s_target) endif;
  }
}

/* Map transition2: map transitions from state to FinalState
  into state to fatherState in lower regions*/
top relation MapTransition2{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
```

```
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::State{},
    target = target_s:uml::FinalState{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    MapState(target_s.container.state,target_t);
    MapRegion2(reg_source,reg_target);
    MapState(source_s,source_t);
    if s_source.trigger->size()=0 then true
      else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then true
      else MapActivityAux(s_source,s_target) endif;
  }
}

/* Map transition3: map transitions from state to
  CompositeState into state to firstPseudostate*/
top relation MapTransition3{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::State{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    target_s.isComposite();
    MapState(source_s,source_t);
    MapPseudoState2(target_s.region->at(1).subvertex->
      select(v :uml::Vertex| v.incoming->isEmpty())->at(1),target_t);
    MapRegion2(reg_source,reg_target) or
```

```
      MapRegion(reg_source,reg_target);
    if s_source.trigger->size()=0 then true
      else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then true
      else MapActivityAux(s_source,s_target) endif;
  }
}

/* Map transition4: map transitions from pseudostate to
   CompositeState into state to firstPseudostate in lower
   Region*/
top relation MapTransition4{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::Pseudostate{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
  when{
    target_s.isComposite();
    MapPseudoState2(source_s,source_t);
    MapPseudoState2(target_s.region->at(1).subvertex->
      select(v :uml::Vertex| v.incoming->isEmpty())->at(1),target_t);
    MapRegion2(reg_source,reg_target) or
      MapRegion(reg_source,reg_target);
    if s_source.trigger->size()=0 then true
      else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then true
      else MapActivityAux(s_source,s_target) endif;
  }
}

/* Map transition5: map transitions from pseudostate to
  CompositeState into pseudostate to firstPseudostate in
  upper Region*/
```

```
top relation MapTransition5{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::Pseudostate{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::Pseudostate{},
    target = target_t:uml::State{}
  };
  when{
    target_s.isComposite();
    MapPseudoState(source_s,source_t);
    MapPseudoState2(target_s.region->at(1).subvertex->
      select(v :uml::Vertex| v.incoming->isEmpty())->at(1),target_t);
    MapRegion2(reg_source,reg_target) or
      MapRegion(reg_source,reg_target);
    if s_source.trigger->size()=0 then true
      else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then true
      else MapActivityAux(s_source,s_target) endif;
  }
}

/* Map transition6: map transitions from Pseudostate to
  simple state in lower regions*/
top relation MapTransition6{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::Pseudostate{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::State{},
    target = target_t:uml::State{}
  };
```

```
  when{
    not target_s.isComposite();
    MapPseudoState2(source_s,source_t) and
      (MapState(target_s,target_t) or
        MapFinalState(target_s,target_t));
    MapRegion2(reg_source,reg_target);
    if s_source.trigger->size()=0 then true
      else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then true
      else MapActivityAux(s_source,s_target) endif;
  }
}

/* Map transition7: map transitions from Pseudostate to
  simple state in upper region*/
top relation MapTransition7{
  ntransition : String;
  checkonly domain  source s_source :uml::Transition {
    name = ntransition,
    container = reg_source:uml::Region{},
    source = source_s:uml::Pseudostate{},
    target = target_s:uml::State{}
  };
  enforce domain target s_target :uml::Transition{
    name = ntransition,
    container = reg_target:uml::Region{},
    source = source_t:uml::Pseudostate{},
    target = target_t:uml::State{}
  };
  when{
    not target_s.isComposite();
    MapPseudoState(source_s,source_t) and
      (MapState(target_s,target_t) or MapFinalState(target_s,target_t));
    MapRegion(reg_source,reg_target);
    if s_source.trigger->size()=0 then true
      else MapTriggerAux(s_source,s_target) endif;
  }
  where{
    if s_source.effect.oclIsUndefined() then true
      else MapActivityAux(s_source,s_target) endif;
  }
}

query isFinal(s : uml::State) : Boolean{
```

```
    if(s.outgoing->isEmpty()) then
      true
    else
      false
    endif
  }
}
```

Appendix B

Implementación Código Java para ejecutar transformaciones definidas en Medini QVT-Relational

Código completo de la implementación del metamodelo DEVS y de la inicialización del motor de Medini QVT-Relations para poder ejecutar las transformaciones.

1. Implementación del metamodelo DEVS.
   Interface AtomicModel.java.

```
/**
*/
package devs;

import org.eclipse.emf.common.util.EList;

import org.eclipse.emf.ecore.EObject;

/**
* <!-- begin-user-doc -->
* A representation of the model object '<em><b>Atomic Model</b></em>'.
* <!-- end-user-doc -->
*
* <p>
* The following features are supported:
* <ul>
*   <li>{@link devs.AtomicModel#getName <em>Name</em>}</li>
*   <li>{@link devs.AtomicModel#getState <em>State</em>}</li>
```

```
*   <li>{@link devs.AtomicModel#getEvent <em>Event</em>}</li>
*   <li>{@link devs.AtomicModel#getTransition <em>Transition</em>}</li>
*   <li>{@link devs.AtomicModel#getOutputFunction <em>Output Function</em>}</li>
* </ul>
* </p>
*
* @see devs.DevsPackage#getAtomicModel()
* @model
* @generated
*/
public interface AtomicModel extends EObject {
  /**
  * Returns the value of the '<em><b>Name</b></em>' attribute.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Name</em>' attribute isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Name</em>' attribute.
  * @see #setName(String)
  * @see devs.DevsPackage#getAtomicModel_Name()
  * @model
  * @generated
  */
  String getName();

  /**
  * Sets the value of the '{@link devs.AtomicModel#getName
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @param value the new value of the '<em>Name</em>' attribute.
  * @see #getName()
  * @generated
  */
  void setName(String value);

  /**
  * Returns the value of the '<em><b>State</b></em>'
  * containment reference list.
  * The list contents are of type {@link devs.State}.
  * It is bidirectional and its opposite is '{@link
  * devs.State#getAtomicModel <em>Atomic Model</em>}'.
  * <!-- begin-user-doc -->
  * <p>
```

```
* If the meaning of the '<em>State</em>' containment
* reference list isn't clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>State</em>' containment
* reference list.
* @see devs.DevsPackage#getAtomicModel_State()
* @see devs.State#getAtomicModel
* @model opposite="atomicModel" containment="true"
* required="true"
* @generated
*/
EList<State> getState();

/**
* Returns the value of the '<em><b>Event</b></em>'
* containment reference list.
* The list contents are of type {@link devs.Event}.
* It is bidirectional and its opposite is '{@link
* devs.Event#getAtomicModel <em>Atomic Model</em>}'.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>Event</em>' containment
* reference list isn't clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Event</em>' containment
* reference list.
* @see devs.DevsPackage#getAtomicModel_Event()
* @see devs.Event#getAtomicModel
* @model opposite="atomicModel" containment="true"
* @generated
*/
EList<Event> getEvent();

/**
* Returns the value of the '<em><b>Transition</b></em>'
* containment reference list.
* The list contents are of type {@link devs.Transition}.
* It is bidirectional and its opposite is '{@link
* devs.Transition#getAtomicModel <em>Atomic Model</em>}'.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>Transition</em>'
```

```
  * containment reference list isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Transition</em>'
  * containment reference list.
  * @see devs.DevsPackage#getAtomicModel_Transition()
  * @see devs.Transition#getAtomicModel
  * @model opposite="atomicModel" containment="true"
  * @generated
  */
  EList<Transition> getTransition();

  /**
  * Returns the value of the '<em><b>Output
  * Function</b></em>' containment reference list.
  * The list contents are of type {@link
  * devs.OutputFunction}.
  * It is bidirectional and its opposite is '{@link
  * devs.OutputFunction#getAtomicModel <em>Atomic
  * Model</em>}'.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Output Function</em>'
  * containment reference list isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Output Function</em>'
  * containment reference list.
  * @see devs.DevsPackage#getAtomicModel_OutputFunction()
  * @see devs.OutputFunction#getAtomicModel
  * @model opposite="atomicModel" containment="true"
  * @generated
  */
  EList<OutputFunction> getOutputFunction();

} // AtomicModel
```

Interface Event.java.

```
/**
*/
package devs;
```

```
import org.eclipse.emf.ecore.EObject;

/**
* <!-- begin-user-doc -->
* A representation of the model object '<em><b>Event</b></em>'.
* <!-- end-user-doc -->
*
* <p>
* The following features are supported:
* <ul>
*   <li>{@link devs.Event#getName <em>Name</em>}</li>
*   <li>{@link devs.Event#getAtomicModel <em>Atomic
*   Model</em>}</li>
* </ul>
* </p>
*
* @see devs.DevsPackage#getEvent()
* @model
* @generated
*/
public interface Event extends EObject {
  /**
  * Returns the value of the '<em><b>Name</b></em>'
  * attribute.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Name</em>' attribute isn't
  * clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Name</em>' attribute.
  * @see #setName(String)
  * @see devs.DevsPackage#getEvent_Name()
  * @model
  * @generated
  */
  String getName();

  /**
  * Sets the value of the '{@link devs.Event#getName
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @param value the new value of the '<em>Name</em>'
  * attribute.
```

```
  * @see #getName()
  * @generated
  */
  void setName(String value);

  /**
  * Returns the value of the '<em><b>Atomic
  * Model</b></em>' container reference.
  * It is bidirectional and its opposite is '{@link
  * devs.AtomicModel#getEvent <em>Event</em>}'.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Atomic Model</em>'
  * container reference isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Atomic Model</em>'
  * container reference.
  * @see #setAtomicModel(AtomicModel)
  * @see devs.DevsPackage#getEvent_AtomicModel()
  * @see devs.AtomicModel#getEvent
  * @model opposite="event" required="true"
  * transient="false"
  * @generated
  */
  AtomicModel getAtomicModel();

  /**
  * Sets the value of the '{@link
  * devs.Event#getAtomicModel <em>Atomic Model</em>}'
  * container reference.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @param value the new value of the '<em>Atomic
  * Model</em>' container reference.
  * @see #getAtomicModel()
  * @generated
  */
  void setAtomicModel(AtomicModel value);

} // Event
```

Interface ExternalTransition.java.

```
/**
*/
package devs;


/**
* <!-- begin-user-doc -->
* A representation of the model object '<em><b>External Transition</b></em>'.
* <!-- end-user-doc -->
*
* <p>
* The following features are supported:
* <ul>
*   <li>{@link devs.ExternalTransition#getInputEvent <em>Input Event</em>}</li>
* </ul>
* </p>
*
* @see devs.DevsPackage#getExternalTransition()
* @model
* @generated
*/
public interface ExternalTransition extends Transition {
  /**
  * Returns the value of the '<em><b>Input Event</b></em>'
  * reference.
  * It is bidirectional and its opposite is '{@link
  * devs.InputEvent#getExternalTransition <em>External
  * Transition</em>}'.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Input Event</em>' reference
  * isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Input Event</em>'
  * reference.
  * @see #setInputEvent(InputEvent)
  * @see
  * devs.DevsPackage#getExternalTransition_InputEvent()
  * @see devs.InputEvent#getExternalTransition
  * @model opposite="externalTransition" required="true"
  * @generated
  */
  InputEvent getInputEvent();
```

```
  /**
  * Sets the value of the '{@link
  * devs.ExternalTransition#getInputEvent <em>Input
  * Event</em>}' reference.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @param value the new value of the '<em>Input
  * Event</em>' reference.
  * @see #getInputEvent()
  * @generated
  */
  void setInputEvent(InputEvent value);

} // ExternalTransition
```

Interface InputEvent.java.

```
/**
*/
package devs;

import org.eclipse.emf.common.util.EList;

/**
* <!-- begin-user-doc -->
* A representation of the model object '<em><b>Input
* Event</b></em>'.
* <!-- end-user-doc -->
*
* <p>
* The following features are supported:
* <ul>
*   <li>{@link devs.InputEvent#getExternalTransition
* <em>External Transition</em>}</li>
* </ul>
* </p>
*
* @see devs.DevsPackage#getInputEvent()
* @model
* @generated
*/
public interface InputEvent extends Event {
  /**
  * Returns the value of the '<em><b>External
  * Transition</b></em>' reference list.
```

```
  * The list contents are of type {@link
  * devs.ExternalTransition}.
  * It is bidirectional and its opposite is '{@link
  * devs.ExternalTransition#getInputEvent <em>Input
  * Event</em>}'.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>External Transition</em>'
  * reference list isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>External
  * Transition</em>' reference list.
  * @see
  * devs.DevsPackage#getInputEvent_ExternalTransition()
  * @see devs.ExternalTransition#getInputEvent
  * @model opposite="inputEvent"
  * @generated
  */
  EList<ExternalTransition> getExternalTransition();

} // InputEvent
```

Interface InternalTransition.java.

```
/**
*/
package devs;


/**
* <!-- begin-user-doc -->
* A representation of the model object '<em><b>Internal Transition</b></em>'.
* <!-- end-user-doc -->
*
*
* @see devs.DevsPackage#getInternalTransition()
* @model
* @generated
*/
public interface InternalTransition extends Transition {
} // InternalTransition
```

Interface OutputEvent.java.

```
/**
*/
package devs;

import org.eclipse.emf.common.util.EList;

/**
* <!-- begin-user-doc -->
* A representation of the model object '<em><b>Output
* Event</b></em>'.
* <!-- end-user-doc -->
*
* <p>
* The following features are supported:
* <ul>
*   <li>{@link devs.OutputEvent#getOutputFunction
* <em>Output Function</em>}</li>
* </ul>
* </p>
*
* @see devs.DevsPackage#getOutputEvent()
* @model
* @generated
*/
public interface OutputEvent extends Event {
  /**
  * Returns the value of the '<em><b>Output
  * Function</b></em>' reference list.
  * The list contents are of type {@link
  * devs.OutputFunction}.
  * It is bidirectional and its opposite is '{@link
  * devs.OutputFunction#getOutputEvent <em>Output
  * Event</em>}'.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Output Function</em>'
  * reference list isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Output Function</em>'
  * reference list.
  * @see devs.DevsPackage#getOutputEvent_OutputFunction()
  * @see devs.OutputFunction#getOutputEvent
  * @model opposite="outputEvent"
  * @generated
```

```
  */
  EList<OutputFunction> getOutputFunction();

} // OutputEvent
```

Interface OutputFunction.java.

```
/**
*/
package devs;

import org.eclipse.emf.ecore.EObject;

/**
* <!-- begin-user-doc -->
* A representation of the model object '<em><b>Output Function</b></em>'.
* <!-- end-user-doc -->
*
* <p>
* The following features are supported:
* <ul>
*   <li>{@link devs.OutputFunction#getName
* <em>Name</em>}</li>
*   <li>{@link devs.OutputFunction#getSource
* <em>Source</em>}</li>
*   <li>{@link devs.OutputFunction#getOutputEvent
* <em>Output Event</em>}</li>
*   <li>{@link devs.OutputFunction#getAtomicModel
* <em>Atomic Model</em>}</li>
* </ul>
* </p>
*
* @see devs.DevsPackage#getOutputFunction()
* @model
* @generated
*/
public interface OutputFunction extends EObject {
  /**
  * Returns the value of the '<em><b>Name</b></em>'
  * attribute.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Name</em>' attribute isn't
  * clear,
  * there really should be more of a description here...
```

```
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Name</em>' attribute.
* @see #setName(String)
* @see devs.DevsPackage#getOutputFunction_Name()
* @model
* @generated
*/
String getName();

/**
* Sets the value of the '{@link
* devs.OutputFunction#getName <em>Name</em>}' attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @param value the new value of the '<em>Name</em>'
* attribute.
* @see #getName()
* @generated
*/
void setName(String value);

/**
* Returns the value of the '<em><b>Source</b></em>'
* reference.
* It is bidirectional and its opposite is '{@link
* devs.State#getOutF <em>Out F</em>}'.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>Source</em>' reference
* isn't clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Source</em>' reference.
* @see #setSource(State)
* @see devs.DevsPackage#getOutputFunction_Source()
* @see devs.State#getOutF
* @model opposite="outF" required="true"
* @generated
*/
State getSource();

/**
* Sets the value of the '{@link
* devs.OutputFunction#getSource <em>Source</em>}'
```

```
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @param value the new value of the '<em>Source</em>'
* reference.
* @see #getSource()
* @generated
*/
void setSource(State value);

/**
* Returns the value of the '<em><b>Output
* Event</b></em>' reference.
* It is bidirectional and its opposite is '{@link
* devs.OutputEvent#getOutputFunction <em>Output
* Function</em>}'.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>Output Event</em>'
* reference isn't clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Output Event</em>'
* reference.
* @see #setOutputEvent(OutputEvent)
* @see devs.DevsPackage#getOutputFunction_OutputEvent()
* @see devs.OutputEvent#getOutputFunction
* @model opposite="outputFunction" required="true"
* @generated
*/
OutputEvent getOutputEvent();

/**
* Sets the value of the '{@link
* devs.OutputFunction#getOutputEvent <em>Output
* Event</em>}' reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @param value the new value of the '<em>Output
* Event</em>' reference.
* @see #getOutputEvent()
* @generated
*/
void setOutputEvent(OutputEvent value);
```

```
  /**
  * Returns the value of the '<em><b>Atomic
  * Model</b></em>' container reference.
  * It is bidirectional and its opposite is '{@link
  * devs.AtomicModel#getOutputFunction <em>Output
  * Function</em>}'.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Atomic Model</em>'
  * container reference isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Atomic Model</em>'
  * container reference.
  * @see #setAtomicModel(AtomicModel)
  * @see devs.DevsPackage#getOutputFunction_AtomicModel()
  * @see devs.AtomicModel#getOutputFunction
  * @model opposite="outputFunction" required="true"
  * transient="false"
  * @generated
  */
  AtomicModel getAtomicModel();

  /**
  * Sets the value of the '{@link
  * devs.OutputFunction#getAtomicModel <em>Atomic
  * Model</em>}' container reference.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @param value the new value of the '<em>Atomic
  * Model</em>' container reference.
  * @see #getAtomicModel()
  * @generated
  */
  void setAtomicModel(AtomicModel value);

} // OutputFunction
```

Interface State.java.

```
/**
*/
package devs;
```

```
import org.eclipse.emf.common.util.EList;

import org.eclipse.emf.ecore.EObject;

/**
* <!-- begin-user-doc -->
* A representation of the model object '<em><b>State</b></em>'.
* <!-- end-user-doc -->
*
* <p>
* The following features are supported:
* <ul>
*   <li>{@link devs.State#getLifeTime <em>Life
* Time</em>}</li>
*   <li>{@link devs.State#getName <em>Name</em>}</li>
*   <li>{@link devs.State#getIn <em>In</em>}</li>
*   <li>{@link devs.State#getOut <em>Out</em>}</li>
*   <li>{@link devs.State#getOutF <em>Out F</em>}</li>
*   <li>{@link devs.State#getAtomicModel <em>Atomic
* Model</em>}</li>
* </ul>
* </p>
*
* @see devs.DevsPackage#getState()
* @model
* @generated
*/
public interface State extends EObject {
  /**
  * Returns the value of the '<em><b>Life Time</b></em>'
  * attribute.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Life Time</em>' attribute
  * isn't clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Life Time</em>'
  * attribute.
  * @see #setLifeTime(double)
  * @see devs.DevsPackage#getState_LifeTime()
  * @model
  * @generated
  */
  double getLifeTime();
```

```
/**
* Sets the value of the '{@link devs.State#getLifeTime
* <em>Life Time</em>}' attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @param value the new value of the '<em>Life Time</em>'
* attribute.
* @see #getLifeTime()
* @generated
*/
void setLifeTime(double value);

/**
* Returns the value of the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>Name</em>' attribute isn't
* clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Name</em>' attribute.
* @see #setName(String)
* @see devs.DevsPackage#getState_Name()
* @model
* @generated
*/
String getName();

/**
* Sets the value of the '{@link devs.State#getName
* <em>Name</em>}' attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @param value the new value of the '<em>Name</em>'
* attribute.
* @see #getName()
* @generated
*/
void setName(String value);

/**
* Returns the value of the '<em><b>In</b></em>'
* reference list.
```

```
* The list contents are of type {@link devs.Transition}.
* It is bidirectional and its opposite is '{@link
* devs.Transition#getTarget <em>Target</em>}'.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>In</em>' reference list
* isn't clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>In</em>' reference list.
* @see devs.DevsPackage#getState_In()
* @see devs.Transition#getTarget
* @model opposite="target"
* @generated
*/
EList<Transition> getIn();

/**
* Returns the value of the '<em><b>Out</b></em>'
* reference list.
* The list contents are of type {@link devs.Transition}.
* It is bidirectional and its opposite is '{@link
* devs.Transition#getSource <em>Source</em>}'.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>Out</em>' reference list
* isn't clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Out</em>' reference list.
* @see devs.DevsPackage#getState_Out()
* @see devs.Transition#getSource
* @model opposite="source"
* @generated
*/
EList<Transition> getOut();

/**
* Returns the value of the '<em><b>Out F</b></em>'
* reference.
* It is bidirectional and its opposite is '{@link
* devs.OutputFunction#getSource <em>Source</em>}'.
* <!-- begin-user-doc -->
* <p>
```

```
* If the meaning of the '<em>Out F</em>' reference isn't
* clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Out F</em>' reference.
* @see #setOutF(OutputFunction)
* @see devs.DevsPackage#getState_OutF()
* @see devs.OutputFunction#getSource
* @model opposite="source" required="true"
* @generated
*/
OutputFunction getOutF();

/**
* Sets the value of the '{@link devs.State#getOutF
* <em>Out F</em>}' reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @param value the new value of the '<em>Out F</em>'
* reference.
* @see #getOutF()
* @generated
*/
void setOutF(OutputFunction value);

/**
* Returns the value of the '<em><b>Atomic
* Model</b></em>' container reference.
* It is bidirectional and its opposite is '{@link
* devs.AtomicModel#getState <em>State</em>}'.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>Atomic Model</em>'
* container reference isn't clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Atomic Model</em>'
* container reference.
* @see #setAtomicModel(AtomicModel)
* @see devs.DevsPackage#getState_AtomicModel()
* @see devs.AtomicModel#getState
* @model opposite="state" required="true"
* transient="false"
* @generated
```

```
  */
  AtomicModel getAtomicModel();

  /**
  * Sets the value of the '{@link
  * devs.State#getAtomicModel <em>Atomic Model</em>}'
  * container reference.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @param value the new value of the '<em>Atomic
  * Model</em>' container reference.
  * @see #getAtomicModel()
  * @generated
  */
  void setAtomicModel(AtomicModel value);

} // State
```

Interface Transition.java.

```
/**
*/
package devs;

import org.eclipse.emf.ecore.EObject;

/**
* <!-- begin-user-doc -->
* A representation of the model object
* '<em><b>Transition</b></em>'.
* <!-- end-user-doc -->
*
* <p>
* The following features are supported:
* <ul>
*   <li>{@link devs.Transition#getName <em>Name</em>}</li>
*   <li>{@link devs.Transition#getSource
* <em>Source</em>}</li>
*   <li>{@link devs.Transition#getTarget
* <em>Target</em>}</li>
*   <li>{@link devs.Transition#getAtomicModel <em>Atomic
* Model</em>}</li>
* </ul>
* </p>
*
```

```
* @see devs.DevsPackage#getTransition()
* @model
* @generated
*/
public interface Transition extends EObject {
  /**
  * Returns the value of the '<em><b>Name</b></em>'
  * attribute.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Name</em>' attribute isn't
  * clear,
  * there really should be more of a description here...
  * </p>
  * <!-- end-user-doc -->
  * @return the value of the '<em>Name</em>' attribute.
  * @see #setName(String)
  * @see devs.DevsPackage#getTransition_Name()
  * @model
  * @generated
  */
  String getName();

  /**
  * Sets the value of the '{@link devs.Transition#getName
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @param value the new value of the '<em>Name</em>'
  * attribute.
  * @see #getName()
  * @generated
  */
  void setName(String value);

  /**
  * Returns the value of the '<em><b>Source</b></em>'
  * reference.
  * It is bidirectional and its opposite is '{@link
  * devs.State#getOut <em>Out</em>}'.
  * <!-- begin-user-doc -->
  * <p>
  * If the meaning of the '<em>Source</em>' reference
  * isn't clear,
  * there really should be more of a description here...
  * </p>
```

```
* <!-- end-user-doc -->
* @return the value of the '<em>Source</em>' reference.
* @see #setSource(State)
* @see devs.DevsPackage#getTransition_Source()
* @see devs.State#getOut
* @model opposite="out" required="true"
* @generated
*/
State getSource();

/**
* Sets the value of the '{@link
* devs.Transition#getSource <em>Source</em>}' reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @param value the new value of the '<em>Source</em>'
* reference.
* @see #getSource()
* @generated
*/
void setSource(State value);

/**
* Returns the value of the '<em><b>Target</b></em>'
* reference.
* It is bidirectional and its opposite is '{@link
* devs.State#getIn <em>In</em>}'.
* <!-- begin-user-doc -->
* <p>
* If the meaning of the '<em>Target</em>' reference
* isn't clear,
* there really should be more of a description here...
* </p>
* <!-- end-user-doc -->
* @return the value of the '<em>Target</em>' reference.
* @see #setTarget(State)
* @see devs.DevsPackage#getTransition_Target()
* @see devs.State#getIn
* @model opposite="in" required="true"
* @generated
*/
State getTarget();

/**
* Sets the value of the '{@link
* devs.Transition#getTarget <em>Target</em>}' reference.
```

```
   * <!-- begin-user-doc -->
   * <!-- end-user-doc -->
   * @param value the new value of the '<em>Target</em>'
   * reference.
   * @see #getTarget()
   * @generated
   */
   void setTarget(State value);

   /**
   * Returns the value of the '<em><b>Atomic
   * Model</b></em>' container reference.
   * It is bidirectional and its opposite is '{@link
   * devs.AtomicModel#getTransition <em>Transition</em>}'.
   * <!-- begin-user-doc -->
   * <p>
   * If the meaning of the '<em>Atomic Model</em>'
   * container reference isn't clear,
   * there really should be more of a description here...
   * </p>
   * <!-- end-user-doc -->
   * @return the value of the '<em>Atomic Model</em>'
   * container reference.
   * @see #setAtomicModel(AtomicModel)
   * @see devs.DevsPackage#getTransition_AtomicModel()
   * @see devs.AtomicModel#getTransition
   * @model opposite="transition" required="true"
   * transient="false"
   * @generated
   */
   AtomicModel getAtomicModel();

   /**
   * Sets the value of the '{@link
   * devs.Transition#getAtomicModel <em>Atomic Model</em>}'
   * container reference.
   * <!-- begin-user-doc -->
   * <!-- end-user-doc -->
   * @param value the new value of the '<em>Atomic
   * Model</em>' container reference.
   * @see #getAtomicModel()
   * @generated
   */
   void setAtomicModel(AtomicModel value);

} // Transition
```

Interface DevsFactory.java.

```
/**
*/
package devs;

import org.eclipse.emf.ecore.EFactory;

/**
* <!-- begin-user-doc -->
* The <b>Factory</b> for the model.
* It provides a create method for each non-abstract class of the model.
* <!-- end-user-doc -->
* @see devs.DevsPackage
* @generated
*/
public interface DevsFactory extends EFactory {
  /**
  * The singleton instance of the factory.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  DevsFactory eINSTANCE = devs.impl.DevsFactoryImpl.init();

  /**
  * Returns a new object of class '<em>Atomic Model</em>'.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @return a new object of class '<em>Atomic Model</em>'.
  * @generated
  */
  AtomicModel createAtomicModel();

  /**
  * Returns a new object of class '<em>State</em>'.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @return a new object of class '<em>State</em>'.
  * @generated
  */
  State createState();

  /**
```

```
* Returns a new object of class '<em>Event</em>'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return a new object of class '<em>Event</em>'.
* @generated
*/
Event createEvent();

/**
* Returns a new object of class '<em>Input Event</em>'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return a new object of class '<em>Input Event</em>'.
* @generated
*/
InputEvent createInputEvent();

/**
* Returns a new object of class '<em>Output Event</em>'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return a new object of class '<em>Output Event</em>'.
* @generated
*/
OutputEvent createOutputEvent();

/**
* Returns a new object of class '<em>Transition</em>'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return a new object of class '<em>Transition</em>'.
* @generated
*/
Transition createTransition();

/**
* Returns a new object of class '<em>Internal
* Transition</em>'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return a new object of class '<em>Internal
* Transition</em>'.
* @generated
*/
InternalTransition createInternalTransition();
```

```
  /**
  * Returns a new object of class '<em>External
  * Transition</em>'.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @return a new object of class '<em>External
  * Transition</em>'.
  * @generated
  */
  ExternalTransition createExternalTransition();

  /**
  * Returns a new object of class '<em>Output
  * Function</em>'.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @return a new object of class '<em>Output
  * Function</em>'.
  * @generated
  */
  OutputFunction createOutputFunction();

  /**
  * Returns the package supported by this factory.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @return the package supported by this factory.
  * @generated
  */
  DevsPackage getDevsPackage();

} //DevsFactory
```

Interface DevsPackage.java.

```
/**
*/
package devs;

import org.eclipse.emf.ecore.EAttribute;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EPackage;
import org.eclipse.emf.ecore.EReference;

/**
```

```
* <!-- begin-user-doc -->
* The <b>Package</b> for the model.
* It contains accessors for the meta objects to represent
* <ul>
*   <li>each class,</li>
*   <li>each feature of each class,</li>
*   <li>each operation of each class,</li>
*   <li>each enum,</li>
*   <li>and each data type</li>
* </ul>
* <!-- end-user-doc -->
* @see devs.DevsFactory
* @model kind="package"
* @generated
*/
public interface DevsPackage extends EPackage {
  /**
  * The package name.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  String eNAME = "devs";

  /**
  * The package namespace URI.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  String eNS_URI = "urn:devs.ecore";

  /**
  * The package namespace name.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  String eNS_PREFIX = "devs";

  /**
  * The singleton instance of the package.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
```

```
DevsPackage eINSTANCE = devs.impl.DevsPackageImpl.init();

/**
* The meta object id for the '{@link
* devs.impl.AtomicModelImpl <em>Atomic Model</em>}'
* class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.AtomicModelImpl
* @see devs.impl.DevsPackageImpl#getAtomicModel()
* @generated
*/
int ATOMIC_MODEL = 0;

/**
* The feature id for the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int ATOMIC_MODEL__NAME = 0;

/**
* The feature id for the '<em><b>State</b></em>'
* containment reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
   * @generated
* @ordered
*/
int ATOMIC_MODEL__STATE = 1;

/**
* The feature id for the '<em><b>Event</b></em>'
* containment reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int ATOMIC_MODEL__EVENT = 2;

/**
* The feature id for the '<em><b>Transition</b></em>'
```

```
* containment reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int ATOMIC_MODEL__TRANSITION = 3;

/**
* The feature id for the '<em><b>Output
* Function</b></em>' containment reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int ATOMIC_MODEL__OUTPUT_FUNCTION = 4;

/**
* The number of structural features of the '<em>Atomic
* Model</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int ATOMIC_MODEL_FEATURE_COUNT = 5;

/**
* The number of operations of the '<em>Atomic
* Model</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int ATOMIC_MODEL_OPERATION_COUNT = 0;

/**
* The meta object id for the '{@link devs.impl.StateImpl
* <em>State</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.StateImpl
* @see devs.impl.DevsPackageImpl#getState()
* @generated
```

```
*/
int STATE = 1;

/**
* The feature id for the '<em><b>Life Time</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int STATE__LIFE_TIME = 0;

/**
* The feature id for the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int STATE__NAME = 1;

/**
* The feature id for the '<em><b>In</b></em>' reference
* list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int STATE__IN = 2;

/**
* The feature id for the '<em><b>Out</b></em>' reference
* list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int STATE__OUT = 3;

/**
* The feature id for the '<em><b>Out F</b></em>'
* reference.
```

```
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int STATE__OUT_F = 4;

/**
* The feature id for the '<em><b>Atomic Model</b></em>'
* container reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int STATE__ATOMIC_MODEL = 5;

/**
* The number of structural features of the
* '<em>State</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int STATE_FEATURE_COUNT = 6;

/**
* The number of operations of the '<em>State</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int STATE_OPERATION_COUNT = 0;

/**
* The meta object id for the '{@link devs.impl.EventImpl
* <em>Event</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.EventImpl
* @see devs.impl.DevsPackageImpl#getEvent()
* @generated
*/
int EVENT = 2;
```

```
/**
* The feature id for the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EVENT__NAME = 0;

/**
* The feature id for the '<em><b>Atomic Model</b></em>'
* container reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EVENT__ATOMIC_MODEL = 1;

/**
* The number of structural features of the
* '<em>Event</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EVENT_FEATURE_COUNT = 2;

/**
* The number of operations of the '<em>Event</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EVENT_OPERATION_COUNT = 0;

/**
* The meta object id for the '{@link
* devs.impl.InputEventImpl <em>Input Event</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.InputEventImpl
```

```
* @see devs.impl.DevsPackageImpl#getInputEvent()
* @generated
*/
int INPUT_EVENT = 3;

/**
* The feature id for the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INPUT_EVENT__NAME = EVENT__NAME;

/**
* The feature id for the '<em><b>Atomic Model</b></em>'
* container reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INPUT_EVENT__ATOMIC_MODEL = EVENT__ATOMIC_MODEL;

/**
* The feature id for the '<em><b>External
* Transition</b></em>' reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INPUT_EVENT__EXTERNAL_TRANSITION = EVENT_FEATURE_COUNT + 0;

/**
* The number of structural features of the '<em>Input
* Event</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INPUT_EVENT_FEATURE_COUNT = EVENT_FEATURE_COUNT + 1;

/**
```

```
* The number of operations of the '<em>Input Event</em>'
* class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INPUT_EVENT_OPERATION_COUNT = EVENT_OPERATION_COUNT + 0;

/**
* The meta object id for the '{@link
* devs.impl.OutputEventImpl <em>Output Event</em>}'
* class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.OutputEventImpl
* @see devs.impl.DevsPackageImpl#getOutputEvent()
* @generated
*/
int OUTPUT_EVENT = 4;

/**
* The feature id for the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int OUTPUT_EVENT__NAME = EVENT__NAME;

/**
* The feature id for the '<em><b>Atomic Model</b></em>'
* container reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int OUTPUT_EVENT__ATOMIC_MODEL = EVENT__ATOMIC_MODEL;

/**
* The feature id for the '<em><b>Output
* Function</b></em>' reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @generated
* @ordered
*/
int OUTPUT_EVENT__OUTPUT_FUNCTION = EVENT_FEATURE_COUNT + 0;

/**
* The number of structural features of the '<em>Output
* Event</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int OUTPUT_EVENT_FEATURE_COUNT = EVENT_FEATURE_COUNT + 1;

/**
* The number of operations of the '<em>Output
* Event</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
  */
int OUTPUT_EVENT_OPERATION_COUNT = EVENT_OPERATION_COUNT + 0;

/**
* The meta object id for the '{@link
* devs.impl.TransitionImpl <em>Transition</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.TransitionImpl
* @see devs.impl.DevsPackageImpl#getTransition()
* @generated
*/
int TRANSITION = 5;

/**
* The feature id for the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int TRANSITION__NAME = 0;
```

```
/**
* The feature id for the '<em><b>Source</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int TRANSITION__SOURCE = 1;

/**
* The feature id for the '<em><b>Target</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int TRANSITION__TARGET = 2;

/**
* The feature id for the '<em><b>Atomic Model</b></em>'
* container reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int TRANSITION__ATOMIC_MODEL = 3;

/**
* The number of structural features of the
* '<em>Transition</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
  */
int TRANSITION_FEATURE_COUNT = 4;

/**
* The number of operations of the '<em>Transition</em>'
* class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
```

```
* @ordered
*/
int TRANSITION_OPERATION_COUNT = 0;

/**
* The meta object id for the '{@link
* devs.impl.InternalTransitionImpl <em>Internal
* Transition</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.InternalTransitionImpl
* @see devs.impl.DevsPackageImpl#getInternalTransition()
* @generated
*/
int INTERNAL_TRANSITION = 6;

/**
* The feature id for the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INTERNAL_TRANSITION__NAME = TRANSITION__NAME;

/**
* The feature id for the '<em><b>Source</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INTERNAL_TRANSITION__SOURCE = TRANSITION__SOURCE;

/**
* The feature id for the '<em><b>Target</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INTERNAL_TRANSITION__TARGET = TRANSITION__TARGET;
```

```
/**
* The feature id for the '<em><b>Atomic Model</b></em>'
* container reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INTERNAL_TRANSITION__ATOMIC_MODEL = TRANSITION__ATOMIC_MODEL;

/**
* The number of structural features of the '<em>Internal
* Transition</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INTERNAL_TRANSITION_FEATURE_COUNT =
  TRANSITION_FEATURE_COUNT + 0;

/**
* The number of operations of the '<em>Internal
* Transition</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int INTERNAL_TRANSITION_OPERATION_COUNT =
  TRANSITION_OPERATION_COUNT + 0;

/**
* The meta object id for the '{@link
* devs.impl.ExternalTransitionImpl <em>External
* Transition</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.ExternalTransitionImpl
* @see devs.impl.DevsPackageImpl#getExternalTransition()
* @generated
*/
int EXTERNAL_TRANSITION = 7;

/**
* The feature id for the '<em><b>Name</b></em>'
```

```
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EXTERNAL_TRANSITION__NAME = TRANSITION__NAME;

/**
* The feature id for the '<em><b>Source</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EXTERNAL_TRANSITION__SOURCE = TRANSITION__SOURCE;

/**
* The feature id for the '<em><b>Target</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EXTERNAL_TRANSITION__TARGET = TRANSITION__TARGET;

/**
* The feature id for the '<em><b>Atomic Model</b></em>'
* container reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EXTERNAL_TRANSITION__ATOMIC_MODEL = TRANSITION__ATOMIC_MODEL;

/**
* The feature id for the '<em><b>Input Event</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
```

```
int EXTERNAL_TRANSITION__INPUT_EVENT =
  TRANSITION_FEATURE_COUNT + 0;

/**
* The number of structural features of the '<em>External
* Transition</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EXTERNAL_TRANSITION_FEATURE_COUNT =
  TRANSITION_FEATURE_COUNT + 1;

/**
* The number of operations of the '<em>External
* Transition</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int EXTERNAL_TRANSITION_OPERATION_COUNT =
 TRANSITION_OPERATION_COUNT + 0;

/**
* The meta object id for the '{@link
* devs.impl.OutputFunctionImpl <em>Output
* Function</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.OutputFunctionImpl
* @see devs.impl.DevsPackageImpl#getOutputFunction()
* @generated
*/
int OUTPUT_FUNCTION = 8;

/**
* The feature id for the '<em><b>Name</b></em>'
* attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int OUTPUT_FUNCTION__NAME = 0;
```

```
/**
* The feature id for the '<em><b>Source</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int OUTPUT_FUNCTION__SOURCE = 1;

/**
* The feature id for the '<em><b>Output Event</b></em>'
* reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int OUTPUT_FUNCTION__OUTPUT_EVENT = 2;

/**
* The feature id for the '<em><b>Atomic Model</b></em>'
* container reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int OUTPUT_FUNCTION__ATOMIC_MODEL = 3;

/**
* The number of structural features of the '<em>Output
* Function</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
* @ordered
*/
int OUTPUT_FUNCTION_FEATURE_COUNT = 4;

/**
* The number of operations of the '<em>Output
* Function</em>' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @generated
* @ordered
*/
int OUTPUT_FUNCTION_OPERATION_COUNT = 0;


/**
* Returns the meta object for class '{@link
* devs.AtomicModel <em>Atomic Model</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for class '<em>Atomic
* Model</em>'.
* @see devs.AtomicModel
* @generated
*/
EClass getAtomicModel();

/**
* Returns the meta object for the attribute '{@link
* devs.AtomicModel#getName <em>Name</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the attribute
* '<em>Name</em>'.
* @see devs.AtomicModel#getName()
* @see #getAtomicModel()
* @generated
*/
EAttribute getAtomicModel_Name();

/**
* Returns the meta object for the containment reference
* list '{@link devs.AtomicModel#getState
* <em>State</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the containment reference
* list '<em>State</em>'.
* @see devs.AtomicModel#getState()
* @see #getAtomicModel()
* @generated
*/
EReference getAtomicModel_State();

/**
```

```
* Returns the meta object for the containment reference
* list '{@link devs.AtomicModel#getEvent
* <em>Event</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the containment reference
* list '<em>Event</em>'.
* @see devs.AtomicModel#getEvent()
* @see #getAtomicModel()
* @generated
*/
EReference getAtomicModel_Event();

/**
* Returns the meta object for the containment reference
* list '{@link devs.AtomicModel#getTransition
* <em>Transition</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the containment reference
* list '<em>Transition</em>'.
* @see devs.AtomicModel#getTransition()
* @see #getAtomicModel()
* @generated
*/
EReference getAtomicModel_Transition();

/**
* Returns the meta object for the containment reference
* list '{@link devs.AtomicModel#getOutputFunction
* <em>Output Function</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the containment reference
* list '<em>Output Function</em>'.
* @see devs.AtomicModel#getOutputFunction()
* @see #getAtomicModel()
* @generated
*/
EReference getAtomicModel_OutputFunction();

/**
* Returns the meta object for class '{@link devs.State
* <em>State</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @return the meta object for class '<em>State</em>'.
* @see devs.State
* @generated
*/
EClass getState();

/**
* Returns the meta object for the attribute '{@link
* devs.State#getLifeTime <em>Life Time</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the attribute '<em>Life
* Time</em>'.
* @see devs.State#getLifeTime()
* @see #getState()
* @generated
*/
EAttribute getState_LifeTime();

/**
* Returns the meta object for the attribute '{@link
* devs.State#getName <em>Name</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the attribute
* '<em>Name</em>'.
* @see devs.State#getName()
* @see #getState()
* @generated
*/
EAttribute getState_Name();

/**
* Returns the meta object for the reference list '{@link
* devs.State#getIn <em>In</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the reference list
* '<em>In</em>'.
* @see devs.State#getIn()
* @see #getState()
* @generated
*/
EReference getState_In();

/**
```

```
* Returns the meta object for the reference list '{@link
* devs.State#getOut <em>Out</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the reference list
* '<em>Out</em>'.
* @see devs.State#getOut()
* @see #getState()
* @generated
*/
EReference getState_Out();

/**
* Returns the meta object for the reference '{@link
* devs.State#getOutF <em>Out F</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the reference '<em>Out
* F</em>'.
* @see devs.State#getOutF()
* @see #getState()
* @generated
*/
EReference getState_OutF();

/**
* Returns the meta object for the container reference
* '{@link devs.State#getAtomicModel <em>Atomic
* Model</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the container reference
* '<em>Atomic Model</em>'.
* @see devs.State#getAtomicModel()
* @see #getState()
* @generated
*/
EReference getState_AtomicModel();

/**
* Returns the meta object for class '{@link devs.Event
* <em>Event</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for class '<em>Event</em>'.
* @see devs.Event
```

```
* @generated
*/
EClass getEvent();

/**
* Returns the meta object for the attribute '{@link
* devs.Event#getName <em>Name</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the attribute
* '<em>Name</em>'.
* @see devs.Event#getName()
* @see #getEvent()
* @generated
*/
EAttribute getEvent_Name();

/**
* Returns the meta object for the container reference
* '{@link devs.Event#getAtomicModel <em>Atomic
* Model</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the container reference
* '<em>Atomic Model</em>'.
* @see devs.Event#getAtomicModel()
* @see #getEvent()
* @generated
*/
EReference getEvent_AtomicModel();

/**
* Returns the meta object for class '{@link
* devs.InputEvent <em>Input Event</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for class '<em>Input
* Event</em>'.
* @see devs.InputEvent
* @generated
*/
EClass getInputEvent();

/**
* Returns the meta object for the reference list '{@link
* devs.InputEvent#getExternalTransition <em>External
```

```
* Transition</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the reference list
* '<em>External Transition</em>'.
* @see devs.InputEvent#getExternalTransition()
* @see #getInputEvent()
* @generated
*/
EReference getInputEvent_ExternalTransition();

/**
* Returns the meta object for class '{@link
* devs.OutputEvent <em>Output Event</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for class '<em>Output
* Event</em>'.
* @see devs.OutputEvent
* @generated
*/
EClass getOutputEvent();

/**
* Returns the meta object for the reference list '{@link
* devs.OutputEvent#getOutputFunction <em>Output
* Function</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the reference list
* '<em>Output Function</em>'.
* @see devs.OutputEvent#getOutputFunction()
* @see #getOutputEvent()
* @generated
*/
EReference getOutputEvent_OutputFunction();

/**
* Returns the meta object for class '{@link
* devs.Transition <em>Transition</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for class
* '<em>Transition</em>'.
* @see devs.Transition
* @generated
```

```
*/
EClass getTransition();

/**
* Returns the meta object for the attribute '{@link
* devs.Transition#getName <em>Name</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the attribute
* '<em>Name</em>'.
* @see devs.Transition#getName()
* @see #getTransition()
* @generated
*/
EAttribute getTransition_Name();

/**
* Returns the meta object for the reference '{@link
* devs.Transition#getSource <em>Source</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the reference
* '<em>Source</em>'.
* @see devs.Transition#getSource()
* @see #getTransition()
* @generated
*/
EReference getTransition_Source();

/**
* Returns the meta object for the reference '{@link
* devs.Transition#getTarget <em>Target</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the reference
* '<em>Target</em>'.
* @see devs.Transition#getTarget()
* @see #getTransition()
* @generated
*/
EReference getTransition_Target();

/**
* Returns the meta object for the container reference
* '{@link devs.Transition#getAtomicModel <em>Atomic
* Model</em>}'.
```

```
   * <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the container reference
* '<em>Atomic Model</em>'.
* @see devs.Transition#getAtomicModel()
* @see #getTransition()
* @generated
*/
EReference getTransition_AtomicModel();

/**
* Returns the meta object for class '{@link
* devs.InternalTransition <em>Internal Transition</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for class '<em>Internal
* Transition</em>'.
* @see devs.InternalTransition
* @generated
*/
EClass getInternalTransition();

/**
* Returns the meta object for class '{@link
* devs.ExternalTransition <em>External Transition</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for class '<em>External
* Transition</em>'.
* @see devs.ExternalTransition
* @generated
*/
EClass getExternalTransition();

/**
* Returns the meta object for the reference '{@link
* devs.ExternalTransition#getInputEvent <em>Input
* Event</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the reference '<em>Input
* Event</em>'.
* @see devs.ExternalTransition#getInputEvent()
* @see #getExternalTransition()
* @generated
*/
```

```
EReference getExternalTransition_InputEvent();

/**
* Returns the meta object for class '{@link
* devs.OutputFunction <em>Output Function</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for class '<em>Output
* Function</em>'.
* @see devs.OutputFunction
* @generated
*/
EClass getOutputFunction();

/**
* Returns the meta object for the attribute '{@link
* devs.OutputFunction#getName <em>Name</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the attribute
* '<em>Name</em>'.
* @see devs.OutputFunction#getName()
* @see #getOutputFunction()
* @generated
*/
EAttribute getOutputFunction_Name();

/**
* Returns the meta object for the reference '{@link
* devs.OutputFunction#getSource <em>Source</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
   * @return the meta object for the reference
   * '<em>Source</em>'.
* @see devs.OutputFunction#getSource()
* @see #getOutputFunction()
* @generated
*/
EReference getOutputFunction_Source();

/**
* Returns the meta object for the reference '{@link
* devs.OutputFunction#getOutputEvent <em>Output
* Event</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @return the meta object for the reference '<em>Output
* Event</em>'.
* @see devs.OutputFunction#getOutputEvent()
* @see #getOutputFunction()
* @generated
*/
EReference getOutputFunction_OutputEvent();

/**
* Returns the meta object for the container reference
* '{@link devs.OutputFunction#getAtomicModel <em>Atomic
* Model</em>}'.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the meta object for the container reference
* '<em>Atomic Model</em>'.
* @see devs.OutputFunction#getAtomicModel()
* @see #getOutputFunction()
* @generated
*/
EReference getOutputFunction_AtomicModel();

/**
* Returns the factory that creates the instances of the
* model.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the factory that creates the instances of the
* model.
* @generated
*/
DevsFactory getDevsFactory();

/**
* <!-- begin-user-doc -->
* Defines literals for the meta objects that represent
* <ul>
*   <li>each class,</li>
*   <li>each feature of each class,</li>
*   <li>each operation of each class,</li>
*   <li>each enum,</li>
*   <li>and each data type</li>
* </ul>
* <!-- end-user-doc -->
* @generated
*/
```

```
interface Literals {
  /**
  * The meta object literal for the '{@link
  * devs.impl.AtomicModelImpl <em>Atomic Model</em>}'
  * class.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see devs.impl.AtomicModelImpl
  * @see devs.impl.DevsPackageImpl#getAtomicModel()
  * @generated
  */
  EClass ATOMIC_MODEL = eINSTANCE.getAtomicModel();

  /**
  * The meta object literal for the
  * '<em><b>Name</b></em>' attribute feature.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  EAttribute ATOMIC_MODEL__NAME =
    eINSTANCE.getAtomicModel_Name();

  /**
  * The meta object literal for the
  * '<em><b>State</b></em>' containment reference list
  * feature.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  EReference ATOMIC_MODEL__STATE = eINSTANCE.getAtomicModel_State();

  /**
  * The meta object literal for the
  * '<em><b>Event</b></em>' containment reference list
  * feature.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  EReference ATOMIC_MODEL__EVENT =
    eINSTANCE.getAtomicModel_Event();

  /**
  * The meta object literal for the
```

```
* '<em><b>Transition</b></em>' containment reference
* list feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference ATOMIC_MODEL__TRANSITION =
  eINSTANCE.getAtomicModel_Transition();

/**
* The meta object literal for the '<em><b>Output
* Function</b></em>' containment reference list
* feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference ATOMIC_MODEL__OUTPUT_FUNCTION =
  eINSTANCE.getAtomicModel_OutputFunction();

/**
* The meta object literal for the '{@link
* devs.impl.StateImpl <em>State</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.StateImpl
* @see devs.impl.DevsPackageImpl#getState()
* @generated
*/
EClass STATE = eINSTANCE.getState();

/**
* The meta object literal for the '<em><b>Life
* Time</b></em>' attribute feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EAttribute STATE__LIFE_TIME =
  eINSTANCE.getState_LifeTime();

/**
* The meta object literal for the
* '<em><b>Name</b></em>' attribute feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @generated
*/
EAttribute STATE__NAME = eINSTANCE.getState_Name();

/**
* The meta object literal for the '<em><b>In</b></em>'
* reference list feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
  * @generated
*/
EReference STATE__IN = eINSTANCE.getState_In();

/**
* The meta object literal for the
* '<em><b>Out</b></em>' reference list feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference STATE__OUT = eINSTANCE.getState_Out();

/**
* The meta object literal for the '<em><b>Out
* F</b></em>' reference feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference STATE__OUT_F = eINSTANCE.getState_OutF();

/**
* The meta object literal for the '<em><b>Atomic
* Model</b></em>' container reference feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference STATE__ATOMIC_MODEL =
  eINSTANCE.getState_AtomicModel();

/**
* The meta object literal for the '{@link
* devs.impl.EventImpl <em>Event</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @see devs.impl.EventImpl
* @see devs.impl.DevsPackageImpl#getEvent()
* @generated
*/
EClass EVENT = eINSTANCE.getEvent();

/**
* The meta object literal for the
* '<em><b>Name</b></em>' attribute feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EAttribute EVENT__NAME = eINSTANCE.getEvent_Name();

/**
* The meta object literal for the '<em><b>Atomic
* Model</b></em>' container reference feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference EVENT__ATOMIC_MODEL =
  eINSTANCE.getEvent_AtomicModel();

/**
* The meta object literal for the '{@link
* devs.impl.InputEventImpl <em>Input Event</em>}'
* class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.InputEventImpl
* @see devs.impl.DevsPackageImpl#getInputEvent()
* @generated
*/
EClass INPUT_EVENT = eINSTANCE.getInputEvent();

/**
* The meta object literal for the '<em><b>External
* Transition</b></em>' reference list feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference INPUT_EVENT__EXTERNAL_TRANSITION =
  eINSTANCE.getInputEvent_ExternalTransition();
```

```
/**
* The meta object literal for the '{@link
* devs.impl.OutputEventImpl <em>Output Event</em>}'
* class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.OutputEventImpl
* @see devs.impl.DevsPackageImpl#getOutputEvent()
* @generated
*/
EClass OUTPUT_EVENT = eINSTANCE.getOutputEvent();

/**
* The meta object literal for the '<em><b>Output
* Function</b></em>' reference list feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference OUTPUT_EVENT__OUTPUT_FUNCTION =
  eINSTANCE.getOutputEvent_OutputFunction();

/**
* The meta object literal for the '{@link
  devs.impl.TransitionImpl <em>Transition</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.TransitionImpl
* @see devs.impl.DevsPackageImpl#getTransition()
* @generated
*/
EClass TRANSITION = eINSTANCE.getTransition();

/**
* The meta object literal for the
* '<em><b>Name</b></em>' attribute feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EAttribute TRANSITION__NAME =
  eINSTANCE.getTransition_Name();

/**
* The meta object literal for the
```

```
* '<em><b>Source</b></em>' reference feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference TRANSITION__SOURCE =
  eINSTANCE.getTransition_Source();

/**
* The meta object literal for the
* '<em><b>Target</b></em>' reference feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference TRANSITION__TARGET =
  eINSTANCE.getTransition_Target();

/**
* The meta object literal for the '<em><b>Atomic
* Model</b></em>' container reference feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference TRANSITION__ATOMIC_MODEL =
  eINSTANCE.getTransition_AtomicModel();

/**
* The meta object literal for the '{@link
* devs.impl.InternalTransitionImpl <em>Internal
* Transition</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.InternalTransitionImpl
* @see
* devs.impl.DevsPackageImpl#getInternalTransition()
* @generated
*/
EClass INTERNAL_TRANSITION =
  eINSTANCE.getInternalTransition();

/**
* The meta object literal for the '{@link
* devs.impl.ExternalTransitionImpl <em>External
* Transition</em>}' class.
```

```
  * <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.ExternalTransitionImpl
* @see
* devs.impl.DevsPackageImpl#getExternalTransition()
* @generated
*/
EClass EXTERNAL_TRANSITION =
  eINSTANCE.getExternalTransition();

/**
* The meta object literal for the '<em><b>Input
* Event</b></em>' reference feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EReference EXTERNAL_TRANSITION__INPUT_EVENT =
  eINSTANCE.getExternalTransition_InputEvent();

/**
* The meta object literal for the '{@link
* devs.impl.OutputFunctionImpl <em>Output
* Function</em>}' class.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see devs.impl.OutputFunctionImpl
* @see devs.impl.DevsPackageImpl#getOutputFunction()
* @generated
*/
EClass OUTPUT_FUNCTION = eINSTANCE.getOutputFunction();

/**
* The meta object literal for the
* '<em><b>Name</b></em>' attribute feature.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
EAttribute OUTPUT_FUNCTION__NAME =
  eINSTANCE.getOutputFunction_Name();

/**
* The meta object literal for the
* '<em><b>Source</b></em>' reference feature.
* <!-- begin-user-doc -->
```

```
    * <!-- end-user-doc -->
    * @generated
    */
    EReference OUTPUT_FUNCTION__SOURCE =
      eINSTANCE.getOutputFunction_Source();

    /**
    * The meta object literal for the '<em><b>Output
    * Event</b></em>' reference feature.
    * <!-- begin-user-doc -->
    * <!-- end-user-doc -->
    * @generated
    */
    EReference OUTPUT_FUNCTION__OUTPUT_EVENT =
      eINSTANCE.getOutputFunction_OutputEvent();

    /**
    * The meta object literal for the '<em><b>Atomic
    * Model</b></em>' container reference feature.
    * <!-- begin-user-doc -->
    * <!-- end-user-doc -->
    * @generated
    */
    EReference OUTPUT_FUNCTION__ATOMIC_MODEL =
      eINSTANCE.getOutputFunction_AtomicModel();

  }

} //DevsPackage
```

Implementación AtomicModelImpl.java.

```
/**
*/
package devs.impl;

import devs.AtomicModel;
import devs.DevsPackage;
import devs.Event;
import devs.OutputFunction;
import devs.State;
import devs.Transition;

import java.util.Collection;
```

```
import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.notify.NotificationChain;

import org.eclipse.emf.common.util.EList;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.EObjectImpl;

import org.eclipse.emf.ecore.util.EObjectContainmentWithInverseEList;
import org.eclipse.emf.ecore.util.InternalEList;

/**
* <!-- begin-user-doc -->
* An implementation of the model object '<em><b>Atomic
* Model</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link devs.impl.AtomicModelImpl#getName
* <em>Name</em>}</li>
*   <li>{@link devs.impl.AtomicModelImpl#getState
* <em>State</em>}</li>
*   <li>{@link devs.impl.AtomicModelImpl#getEvent
* <em>Event</em>}</li>
*   <li>{@link devs.impl.AtomicModelImpl#getTransition
* <em>Transition</em>}</li>
*   <li>{@link devs.impl.AtomicModelImpl#getOutputFunction
* <em>Output Function</em>}</li>
* </ul>
* </p>
*
* @generated
*/
public class AtomicModelImpl extends EObjectImpl implements AtomicModel {
  /**
  * The default value of the '{@link #getName()
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
  * @ordered
```

```
*/
protected static final String NAME_EDEFAULT = null;

/**
* The cached value of the '{@link #getName()
* <em>Name</em>}' attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getName()
* @generated
* @ordered
*/
protected String name = NAME_EDEFAULT;

/**
* The cached value of the '{@link #getState()
* <em>State</em>}' containment reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getState()
* @generated
* @ordered
*/
protected EList<State> state;

/**
* The cached value of the '{@link #getEvent()
* <em>Event</em>}' containment reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getEvent()
* @generated
* @ordered
*/
protected EList<Event> event;

/**
* The cached value of the '{@link #getTransition()
* <em>Transition</em>}' containment reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getTransition()
* @generated
* @ordered
*/
protected EList<Transition> transition;
```

```
/**
* The cached value of the '{@link #getOutputFunction()
* <em>Output Function</em>}' containment reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getOutputFunction()
* @generated
* @ordered
*/
protected EList<OutputFunction> outputFunction;

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
protected AtomicModelImpl() {
  super();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
protected EClass eStaticClass() {
  return DevsPackage.Literals.ATOMIC_MODEL;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public String getName() {
  return name;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setName(String newName) {
```

```
  String oldName = name;
  name = newName;
  if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
      Notification.SET, DevsPackage.ATOMIC_MODEL__NAME, oldName, name));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EList<State> getState() {
  if (state == null) {
    state = new EObjectContainmentWithInverseEList<State>(State.class,
      this, DevsPackage.ATOMIC_MODEL__STATE, DevsPackage.STATE__ATOMIC_MODEL);
  }
  return state;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EList<Event> getEvent() {
  if (event == null) {
    event = new EObjectContainmentWithInverseEList<Event>(Event.class, this,
      DevsPackage.ATOMIC_MODEL__EVENT, DevsPackage.EVENT__ATOMIC_MODEL);
  }
  return event;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EList<Transition> getTransition() {
  if (transition == null) {
    transition = new EObjectContainmentWithInverseEList<Transition>(Transition.cla
      DevsPackage.ATOMIC_MODEL__TRANSITION, DevsPackage.TRANSITION__ATOMIC_MODEL);
  }
  return transition;
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EList<OutputFunction> getOutputFunction() {
  if (outputFunction == null) {
    outputFunction = new EObjectContainmentWithInverseEList<OutputFunction>(Output
      DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION, DevsPackage.OUTPUT_FUNCTION__ATOM
  }
  return outputFunction;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@SuppressWarnings("unchecked")
@Override
public NotificationChain eInverseAdd(InternalEObject otherEnd,
  int featureID, NotificationChain msgs) {
    switch (featureID) {
      case DevsPackage.ATOMIC_MODEL__STATE:
        return ((InternalEList<InternalEObject>)(InternalEList<?>)getState())
          .basicAdd(otherEnd, msgs);
      case DevsPackage.ATOMIC_MODEL__EVENT:
        return ((InternalEList<InternalEObject>)(InternalEList<?>)getEvent())
          .basicAdd(otherEnd, msgs);
      case DevsPackage.ATOMIC_MODEL__TRANSITION:
        return ((InternalEList<InternalEObject>)(InternalEList<?>)getTransition()
          .basicAdd(otherEnd, msgs);
      case DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION:
        return ((InternalEList<InternalEObject>)(InternalEList<?>)getOutputFunctio
          .basicAdd(otherEnd, msgs);
    }
    return super.eInverseAdd(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseRemove(InternalEObject otherEnd,
```

```
  int featureID, NotificationChain msgs) {
    switch (featureID) {
      case DevsPackage.ATOMIC_MODEL__STATE:
        return ((InternalEList<?>)getState())
          .basicRemove(otherEnd, msgs);
      case DevsPackage.ATOMIC_MODEL__EVENT:
        return ((InternalEList<?>)getEvent())
          .basicRemove(otherEnd, msgs);
      case DevsPackage.ATOMIC_MODEL__TRANSITION:
        return ((InternalEList<?>)getTransition())
          .basicRemove(otherEnd, msgs);
      case DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION:
        return ((InternalEList<?>)getOutputFunction())
          .basicRemove(otherEnd, msgs);
    }
    return super.eInverseRemove(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public Object eGet(int featureID, boolean resolve,
  boolean coreType) {
    switch (featureID) {
      case DevsPackage.ATOMIC_MODEL__NAME:
        return getName();
      case DevsPackage.ATOMIC_MODEL__STATE:
        return getState();
      case DevsPackage.ATOMIC_MODEL__EVENT:
        return getEvent();
      case DevsPackage.ATOMIC_MODEL__TRANSITION:
        return getTransition();
      case DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION:
        return getOutputFunction();
      }
    return super.eGet(featureID, resolve, coreType);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
```

```
@SuppressWarnings("unchecked")
@Override
public void eSet(int featureID, Object newValue) {
  switch (featureID) {
    case DevsPackage.ATOMIC_MODEL__NAME:
      setName((String)newValue);
      return;
    case DevsPackage.ATOMIC_MODEL__STATE:
      getState().clear();
      getState().addAll((Collection<? extends
        State>)newValue);
      return;
    case DevsPackage.ATOMIC_MODEL__EVENT:
      getEvent().clear();
      getEvent().addAll((Collection<? extends
        Event>)newValue);
      return;
    case DevsPackage.ATOMIC_MODEL__TRANSITION:
      getTransition().clear();
      getTransition().addAll((Collection<? extends
        Transition>)newValue);
      return;
    case DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION:
      getOutputFunction().clear();
      getOutputFunction().addAll((Collection<? extends
        OutputFunction>)newValue);
      return;
   }
   super.eSet(featureID, newValue);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
  public void eUnset(int featureID) {
  switch (featureID) {
    case DevsPackage.ATOMIC_MODEL__NAME:
      setName(NAME_EDEFAULT);
      return;
    case DevsPackage.ATOMIC_MODEL__STATE:
      getState().clear();
      return;
    case DevsPackage.ATOMIC_MODEL__EVENT:
```

```
      getEvent().clear();
      return;
    case DevsPackage.ATOMIC_MODEL__TRANSITION:
      getTransition().clear();
      return;
    case DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION:
      getOutputFunction().clear();
      return;
  }
  super.eUnset(featureID);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
   @Override
  public boolean eIsSet(int featureID) {
  switch (featureID) {
    case DevsPackage.ATOMIC_MODEL__NAME:
      return NAME_EDEFAULT == null ? name != null :
        !NAME_EDEFAULT.equals(name);
    case DevsPackage.ATOMIC_MODEL__STATE:
      return state != null && !state.isEmpty();
    case DevsPackage.ATOMIC_MODEL__EVENT:
      return event != null && !event.isEmpty();
    case DevsPackage.ATOMIC_MODEL__TRANSITION:
      return transition != null && !transition.isEmpty();
    case DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION:
      return outputFunction != null &&
        !outputFunction.isEmpty();
  }
  return super.eIsSet(featureID);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public String toString() {
  if (eIsProxy()) return super.toString();

  StringBuffer result = new
```

```
      StringBuffer(super.toString());
    result.append(" (name: ");
    result.append(name);
    result.append(')');
    return result.toString();
  }

} //AtomicModelImpl
```

Implementación EventImpl.java.

```
/**
*/
package devs.impl;

import devs.AtomicModel;
import devs.DevsPackage;
import devs.Event;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.notify.NotificationChain;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.EObjectImpl;

import org.eclipse.emf.ecore.util.EcoreUtil;

/**
* <!-- begin-user-doc -->
* An implementation of the model object
* '<em><b>Event</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link devs.impl.EventImpl#getName
* <em>Name</em>}</li>
*   <li>{@link devs.impl.EventImpl#getAtomicModel
* <em>Atomic Model</em>}</li>
* </ul>
* </p>
*
```

```
* @generated
*/
public class EventImpl extends EObjectImpl implements Event {
  /**
  * The default value of the '{@link #getName()
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
  * @ordered
  */
  protected static final String NAME_EDEFAULT = null;

  /**
  * The cached value of the '{@link #getName()
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
  * @ordered
  */
  protected String name = NAME_EDEFAULT;

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  protected EventImpl() {
    super();
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  protected EClass eStaticClass() {
    return DevsPackage.Literals.EVENT;
  }

  /**
  * <!-- begin-user-doc -->
```

```
* <!-- end-user-doc -->
* @generated
*/
public String getName() {
  return name;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setName(String newName) {
  String oldName = name;
  name = newName;
  if (eNotificationRequired())
  eNotify(new ENotificationImpl(this, Notification.SET,
    DevsPackage.EVENT__NAME, oldName, name));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public AtomicModel getAtomicModel() {
  if (eContainerFeatureID() !=
    DevsPackage.EVENT__ATOMIC_MODEL) return null;
  return (AtomicModel)eInternalContainer();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetAtomicModel(AtomicModel newAtomicModel,
  NotificationChain msgs) {
    msgs = eBasicSetContainer((InternalEObject)newAtomicModel,
      DevsPackage.EVENT__ATOMIC_MODEL, msgs);
    return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @generated
*/
public void setAtomicModel(AtomicModel newAtomicModel) {
  if (newAtomicModel != eInternalContainer() ||
    (eContainerFeatureID() != DevsPackage.EVENT__ATOMIC_MODEL &&
      newAtomicModel != null)) {
    if (EcoreUtil.isAncestor(this, newAtomicModel))
      throw new IllegalArgumentException("Recursive
        containment not allowed for " + toString());
    NotificationChain msgs = null;
    if (eInternalContainer() != null)
      msgs = eBasicRemoveFromContainer(msgs);
    if (newAtomicModel != null)
      msgs = ((InternalEObject)newAtomicModel)
        .eInverseAdd(this, DevsPackage.ATOMIC_MODEL__EVENT,
          AtomicModel.class, msgs);
    msgs = basicSetAtomicModel(newAtomicModel, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
      Notification.SET, DevsPackage.EVENT__ATOMIC_MODEL,
        newAtomicModel, newAtomicModel));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseAdd(InternalEObject otherEnd,
  int featureID, NotificationChain msgs) {
    switch (featureID) {
      case DevsPackage.EVENT__ATOMIC_MODEL:
        if (eInternalContainer() != null)
          msgs = eBasicRemoveFromContainer(msgs);
          return basicSetAtomicModel(
            (AtomicModel)otherEnd, msgs);
    }
    return super.eInverseAdd(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @generated
*/
@Override
public NotificationChain eInverseRemove(InternalEObject
  otherEnd, int featureID, NotificationChain msgs) {
    switch (featureID) {
      case DevsPackage.EVENT__ATOMIC_MODEL:
        return basicSetAtomicModel(null, msgs);
    }
    return super.eInverseRemove(otherEnd,
      featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eBasicRemoveFromContainerFeature(
  NotificationChain msgs) {
    switch (eContainerFeatureID()) {
      case DevsPackage.EVENT__ATOMIC_MODEL:
        return eInternalContainer().eInverseRemove(this,
          DevsPackage.ATOMIC_MODEL__EVENT, AtomicModel.class, msgs);
      }
      return super.eBasicRemoveFromContainerFeature(msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public Object eGet(int featureID, boolean resolve,
  boolean coreType) {
    switch (featureID) {
      case DevsPackage.EVENT__NAME:
        return getName();
      case DevsPackage.EVENT__ATOMIC_MODEL:
        return getAtomicModel();
    }
    return super.eGet(featureID, resolve, coreType);
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public void eSet(int featureID, Object newValue) {
  switch (featureID) {
    case DevsPackage.EVENT__NAME:
      setName((String)newValue);
      return;
    case DevsPackage.EVENT__ATOMIC_MODEL:
      setAtomicModel((AtomicModel)newValue);
      return;
  }
  super.eSet(featureID, newValue);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public void eUnset(int featureID) {
  switch (featureID) {
    case DevsPackage.EVENT__NAME:
      setName(NAME_EDEFAULT);
      return;
    case DevsPackage.EVENT__ATOMIC_MODEL:
      setAtomicModel((AtomicModel)null);
      return;
  }
  super.eUnset(featureID);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public boolean eIsSet(int featureID) {
  switch (featureID) {
    case DevsPackage.EVENT__NAME:
      return NAME_EDEFAULT == null ? name != null :
```

```
          !NAME_EDEFAULT.equals(name);
      case DevsPackage.EVENT__ATOMIC_MODEL:
        return getAtomicModel() != null;
    }
    return super.eIsSet(featureID);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public String toString() {
    if (eIsProxy()) return super.toString();

    StringBuffer result = new
      StringBuffer(super.toString());
    result.append(" (name: ");
    result.append(name);
    result.append(')');
    return result.toString();
  }

} //EventImpl
```

Implementación ExternalTransitionImpl.java.

```
/**
*/
package devs.impl;

import devs.DevsPackage;
import devs.ExternalTransition;
import devs.InputEvent;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.notify.NotificationChain;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.impl.ENotificationImpl;

/**
```

```
* <!-- begin-user-doc -->
* An implementation of the model object '<em><b>External
* Transition</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link
* devs.impl.ExternalTransitionImpl#getInputEvent <em>Input
* Event</em>}</li>
* </ul>
* </p>
*
* @generated
*/
public class ExternalTransitionImpl extends
  TransitionImpl implements ExternalTransition {
    /**
    * The cached value of the '{@link #getInputEvent()
    * <em>Input Event</em>}' reference.
    * <!-- begin-user-doc -->
    * <!-- end-user-doc -->
    * @see #getInputEvent()
    * @generated
    * @ordered
    */
    protected InputEvent inputEvent;

    /**
    * <!-- begin-user-doc -->
    * <!-- end-user-doc -->
    * @generated
    */
    protected ExternalTransitionImpl() {
          super();
    }

    /**
    * <!-- begin-user-doc -->
    * <!-- end-user-doc -->
    * @generated
    */
    @Override
    protected EClass eStaticClass() {
      return DevsPackage.Literals.EXTERNAL_TRANSITION;
    }
```

```
  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  public InputEvent getInputEvent() {
    if (inputEvent != null && inputEvent.eIsProxy()) {
      InternalEObject oldInputEvent =
        (InternalEObject)inputEvent;
      inputEvent =
        (InputEvent)eResolveProxy(oldInputEvent);
      if (inputEvent != oldInputEvent) {
        if (eNotificationRequired())
          eNotify(new ENotificationImpl(this, Notification.RESOLVE,
            DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT,
              oldInputEvent, inputEvent));
      }
    }
    return inputEvent;
  }

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public InputEvent basicGetInputEvent() {
  return inputEvent;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetInputEvent(InputEvent
  newInputEvent, NotificationChain msgs) {
    InputEvent oldInputEvent = inputEvent;
    inputEvent = newInputEvent;
    if (eNotificationRequired()) {
      ENotificationImpl notification = new
        ENotificationImpl(this, Notification.SET,
          DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT,
            oldInputEvent, newInputEvent);
      if (msgs == null) msgs = notification;
```

```
        else msgs.add(notification);
    }
    return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setInputEvent(InputEvent newInputEvent) {
  if (newInputEvent != inputEvent) {
    NotificationChain msgs = null;
    if (inputEvent != null)
      msgs = ((InternalEObject)inputEvent)
        .eInverseRemove(this, DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION,
          InputEvent.class, msgs);
    if (newInputEvent != null)
      msgs = ((InternalEObject)newInputEvent)
        .eInverseAdd(this, DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION,
          InputEvent.class, msgs);
    msgs = basicSetInputEvent(newInputEvent, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
      Notification.SET, DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT,
        newInputEvent, newInputEvent));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseAdd(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT:
      if (inputEvent != null)
        msgs = ((InternalEObject)inputEvent)
         .eInverseRemove(this, DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION,
          InputEvent.class, msgs);
      return basicSetInputEvent((InputEvent)otherEnd, msgs);
  }
```

```
  return super.eInverseAdd(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseRemove(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT:
      return basicSetInputEvent(null, msgs);
  }
  return super.eInverseRemove(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public Object eGet(int featureID, boolean resolve,
 boolean coreType) {
  switch (featureID) {
    case DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT:
      if (resolve) return getInputEvent();
        return basicGetInputEvent();
  }
  return super.eGet(featureID, resolve, coreType);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public void eSet(int featureID, Object newValue) {
  switch (featureID) {
    case DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT:
      setInputEvent((InputEvent)newValue);
      return;
  }
```

```
    super.eSet(featureID, newValue);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public void eUnset(int featureID) {
    switch (featureID) {
    case DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT:
      setInputEvent((InputEvent)null);
      return;
    }
    super.eUnset(featureID);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public boolean eIsSet(int featureID) {
    switch (featureID) {
      case DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT:
        return inputEvent != null;
    }
    return super.eIsSet(featureID);
  }

} //ExternalTransitionImpl
```

Implementación InputEventImpl.java.

```
/**
*/
package devs.impl;

import devs.DevsPackage;
import devs.ExternalTransition;
import devs.InputEvent;

import java.util.Collection;
```

```
import org.eclipse.emf.common.notify.NotificationChain;

import org.eclipse.emf.common.util.EList;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.util.EObjectWithInverseResolvingEList;
import org.eclipse.emf.ecore.util.InternalEList;

/**
* <!-- begin-user-doc -->
* An implementation of the model object '<em><b>Input
* Event</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link
* devs.impl.InputEventImpl#getExternalTransition
* <em>External Transition</em>}</li>
* </ul>
* </p>
*
* @generated
*/
public class InputEventImpl extends
 EventImpl implements InputEvent {
  /**
  * The cached value of the '{@link
  * #getExternalTransition() <em>External
  * Transition</em>}' reference list.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getExternalTransition()
  * @generated
  * @ordered
  */
  protected EList<ExternalTransition> externalTransition;

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
```

```
protected InputEventImpl() {
  super();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
protected EClass eStaticClass() {
  return DevsPackage.Literals.INPUT_EVENT;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EList<ExternalTransition> getExternalTransition() {
  if (externalTransition == null) {
    externalTransition = new
     EObjectWithInverseResolvingEList<ExternalTransition>(ExternalTransition.class
      this, DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION,
       DevsPackage.EXTERNAL_TRANSITION__INPUT_EVENT);
  }
  return externalTransition;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@SuppressWarnings("unchecked")
@Override
public NotificationChain eInverseAdd(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION:
    return ((InternalEList<InternalEObject>)(InternalEList<?>)
     getExternalTransition()).basicAdd(otherEnd, msgs);
  }
  return super.eInverseAdd(otherEnd, featureID, msgs);
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseRemove(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION:
      return ((InternalEList<?>)getExternalTransition())
       .basicRemove(otherEnd, msgs);
  }
  return super.eInverseRemove(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public Object eGet(int featureID, boolean resolve,
 boolean coreType) {
  switch (featureID) {
    case DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION:
      return getExternalTransition();
  }
  return super.eGet(featureID, resolve, coreType);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@SuppressWarnings("unchecked")
@Override
public void eSet(int featureID, Object newValue) {
  switch (featureID) {
    case DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION:
      getExternalTransition().clear();
      getExternalTransition().addAll((Collection<? extends
       ExternalTransition>)newValue);
      return;
  }
```

```
    super.eSet(featureID, newValue);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public void eUnset(int featureID) {
    switch (featureID) {
      case DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION:
        getExternalTransition().clear();
        return;
    }
    super.eUnset(featureID);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public boolean eIsSet(int featureID) {
    switch (featureID) {
      case DevsPackage.INPUT_EVENT__EXTERNAL_TRANSITION:
        return externalTransition != null &&
         !externalTransition.isEmpty();
    }
    return super.eIsSet(featureID);
  }

} //InputEventImpl
```

Implementación InternalTransitionImpl.java.

```
/**
*/
package devs.impl;

import devs.DevsPackage;
import devs.InternalTransition;

import org.eclipse.emf.ecore.EClass;
```

```
/**
* <!-- begin-user-doc -->
* An implementation of the model object '<em><b>Internal
* Transition</b></em>'.
* <!-- end-user-doc -->
* <p>
* </p>
*
* @generated
*/
public class InternalTransitionImpl extends
 TransitionImpl implements InternalTransition {
  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  protected InternalTransitionImpl() {
    super();
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  protected EClass eStaticClass() {
    return DevsPackage.Literals.INTERNAL_TRANSITION;
  }

} //InternalTransitionImpl
```

Implementación OutputEventImpl.java.

```
/**
*/
package devs.impl;

import devs.DevsPackage;
import devs.OutputEvent;
import devs.OutputFunction;

import java.util.Collection;
```

```
import org.eclipse.emf.common.notify.NotificationChain;

import org.eclipse.emf.common.util.EList;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.util.EObjectWithInverseResolvingEList;
import org.eclipse.emf.ecore.util.InternalEList;

/**
* <!-- begin-user-doc -->
* An implementation of the model object '<em><b>Output
* Event</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link devs.impl.OutputEventImpl#getOutputFunction
* <em>Output Function</em>}</li>
* </ul>
* </p>
*
* @generated
*/
public class OutputEventImpl extends
 EventImpl implements OutputEvent {
  /**
  * The cached value of the '{@link #getOutputFunction()
  * <em>Output Function</em>}' reference list.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getOutputFunction()
  * @generated
  * @ordered
  */
  protected EList<OutputFunction> outputFunction;

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  protected OutputEventImpl() {
    super();
```

```
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
protected EClass eStaticClass() {
  return DevsPackage.Literals.OUTPUT_EVENT;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EList<OutputFunction> getOutputFunction() {
  if (outputFunction == null) {
    outputFunction = new EObjectWithInverseResolvingEList<OutputFunction>(
     OutputFunction.class, this, DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION,
      DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT);
  }
  return outputFunction;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@SuppressWarnings("unchecked")
@Override
public NotificationChain eInverseAdd(InternalEObject otherEnd,
 int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION:
    return ((InternalEList<InternalEObject>)(InternalEList<?>)
     getOutputFunction()).basicAdd(otherEnd, msgs);
  }
  return super.eInverseAdd(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @generated
*/
@Override
public NotificationChain eInverseRemove(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION:
    return ((InternalEList<?>)getOutputFunction())
     .basicRemove(otherEnd, msgs);
  }
  return super.eInverseRemove(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public Object eGet(int featureID, boolean resolve,
 boolean coreType) {
  switch (featureID) {
    case DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION:
      return getOutputFunction();
  }
  return super.eGet(featureID, resolve, coreType);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@SuppressWarnings("unchecked")
@Override
public void eSet(int featureID, Object newValue) {
  switch (featureID) {
    case DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION:
      getOutputFunction().clear();
      getOutputFunction().addAll((Collection<? extends
       OutputFunction>)newValue);
        return;
  }
  super.eSet(featureID, newValue);
}
```

```
  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public void eUnset(int featureID) {
    switch (featureID) {
      case DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION:
        getOutputFunction().clear();
           return;
    }
    super.eUnset(featureID);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public boolean eIsSet(int featureID) {
    switch (featureID) {
      case DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION:
        return outputFunction != null &&
         !outputFunction.isEmpty();
    }
    return super.eIsSet(featureID);
  }

} //OutputEventImpl
```

Implementación OutputFunctionImpl.java.

```
/**
*/
package devs.impl;

import devs.AtomicModel;
import devs.DevsPackage;
import devs.OutputEvent;
import devs.OutputFunction;
import devs.State;

import org.eclipse.emf.common.notify.Notification;
```

```
import org.eclipse.emf.common.notify.NotificationChain;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.EObjectImpl;

import org.eclipse.emf.ecore.util.EcoreUtil;

/**
* <!-- begin-user-doc -->
* An implementation of the model object '<em><b>Output
* Function</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link devs.impl.OutputFunctionImpl#getName
* <em>Name</em>}</li>
*   <li>{@link devs.impl.OutputFunctionImpl#getSource
* <em>Source</em>}</li>
*   <li>{@link devs.impl.OutputFunctionImpl#getOutputEvent
* <em>Output Event</em>}</li>
*   <li>{@link devs.impl.OutputFunctionImpl#getAtomicModel
* <em>Atomic Model</em>}</li>
* </ul>
* </p>
*
* @generated
*/
public class OutputFunctionImpl extends EObjectImpl
 implements OutputFunction {
  /**
  * The default value of the '{@link #getName()
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
  * @ordered
  */
  protected static final String NAME_EDEFAULT = null;

  /**
  * The cached value of the '{@link #getName()
```

```
* <em>Name</em>}' attribute.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
  * @see #getName()
* @generated
* @ordered
*/
protected String name = NAME_EDEFAULT;

/**
* The cached value of the '{@link #getSource()
* <em>Source</em>}' reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getSource()
* @generated
* @ordered
*/
protected State source;

/**
* The cached value of the '{@link #getOutputEvent()
* <em>Output Event</em>}' reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getOutputEvent()
* @generated
* @ordered
*/
protected OutputEvent outputEvent;

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
protected OutputFunctionImpl() {
  super();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
```

```
protected EClass eStaticClass() {
  return DevsPackage.Literals.OUTPUT_FUNCTION;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public String getName() {
  return name;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setName(String newName) {
  String oldName = name;
  name = newName;
  if (eNotificationRequired())
    eNotify(new ENotificationImpl(this, Notification.SET,
     DevsPackage.OUTPUT_FUNCTION__NAME, oldName, name));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public State getSource() {
  if (source != null && source.eIsProxy()) {
    InternalEObject oldSource = (InternalEObject)source;
    source = (State)eResolveProxy(oldSource);
    if (source != oldSource) {
      if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, Notification.RESOLVE,
         DevsPackage.OUTPUT_FUNCTION__SOURCE, oldSource, source));
    }
  }
  return source;
}

/**
* <!-- begin-user-doc -->
```

```
* <!-- end-user-doc -->
* @generated
*/
public State basicGetSource() {
  return source;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetSource(State newSource,
 NotificationChain msgs) {
  State oldSource = source;
  source = newSource;
  if (eNotificationRequired()) {
    ENotificationImpl notification = new ENotificationImpl(
     this, Notification.SET, DevsPackage.OUTPUT_FUNCTION__SOURCE, oldSource,
       newSource);
    if (msgs == null) msgs = notification;
     else msgs.add(notification);
  }
  return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setSource(State newSource) {
  if (newSource != source) {
    NotificationChain msgs = null;
    if (source != null)
      msgs = ((InternalEObject)source)
       .eInverseRemove(this, DevsPackage.STATE__OUT_F,
        State.class, msgs);
    if (newSource != null)
      msgs = ((InternalEObject)newSource)
       .eInverseAdd(this, DevsPackage.STATE__OUT_F,
         State.class, msgs);
    msgs = basicSetSource(newSource, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
```

```
    eNotify(new ENotificationImpl(this,
     Notification.SET, DevsPackage.OUTPUT_FUNCTION__SOURCE,
      newSource, newSource));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public OutputEvent getOutputEvent() {
  if (outputEvent != null && outputEvent.eIsProxy()) {
    InternalEObject oldOutputEvent =
     (InternalEObject)outputEvent;
    outputEvent =
     (OutputEvent)eResolveProxy(oldOutputEvent);
    if (outputEvent != oldOutputEvent) {
      if (eNotificationRequired())
       eNotify(new ENotificationImpl(this,
        Notification.RESOLVE, DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT,
         oldOutputEvent, outputEvent));
    }
  }
  return outputEvent;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public OutputEvent basicGetOutputEvent() {
  return outputEvent;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetOutputEvent(OutputEvent
 newOutputEvent, NotificationChain msgs) {
  OutputEvent oldOutputEvent = outputEvent;
  outputEvent = newOutputEvent;
  if (eNotificationRequired()) {
    ENotificationImpl notification =
```

```
      new ENotificationImpl(this, Notification.SET,
       DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT,
        oldOutputEvent, newOutputEvent);
    if (msgs == null) msgs = notification;
     else msgs.add(notification);
  }
  return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setOutputEvent(OutputEvent newOutputEvent) {
  if (newOutputEvent != outputEvent) {
    NotificationChain msgs = null;
    if (outputEvent != null)
      msgs = ((InternalEObject)outputEvent)
       .eInverseRemove(this,
        DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION,
         OutputEvent.class, msgs);
    if (newOutputEvent != null)
      msgs = ((InternalEObject)newOutputEvent)
       .eInverseAdd(this,
        DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION,
         OutputEvent.class, msgs);
    msgs = basicSetOutputEvent(newOutputEvent, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET,
      DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT,
       newOutputEvent, newOutputEvent));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public AtomicModel getAtomicModel() {
  if (eContainerFeatureID() !=
    DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL)
     returnnull;
```

```
  return (AtomicModel)eInternalContainer();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetAtomicModel(AtomicModel
 newAtomicModel, NotificationChain msgs) {
  msgs = eBasicSetContainer((InternalEObject)newAtomicModel,
   DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL, msgs);
  return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setAtomicModel(AtomicModel newAtomicModel) {
  if (newAtomicModel != eInternalContainer() ||
   (eContainerFeatureID() !=
    DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL &&
     newAtomicModel != null)) {
      if (EcoreUtil.isAncestor(this, newAtomicModel))
        throw new IllegalArgumentException("Recursive
         containment not allowed for " + toString());
      NotificationChain msgs = null;
      if (eInternalContainer() != null)
        msgs = eBasicRemoveFromContainer(msgs);
      if (newAtomicModel != null)
        msgs = ((InternalEObject)newAtomicModel)
         .eInverseAdd(this,
          DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION,
           AtomicModel.class, msgs);
      msgs = basicSetAtomicModel(newAtomicModel, msgs);
      if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET,
      DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL,
       newAtomicModel, newAtomicModel));
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseAdd(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.OUTPUT_FUNCTION__SOURCE:
      if (source != null)
        msgs = ((InternalEObject)source)
         .eInverseRemove(this, DevsPackage.STATE__OUT_F,
           State.class, msgs);
      return basicSetSource((State)otherEnd, msgs);
    case DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT:
      if (outputEvent != null)
        msgs = ((InternalEObject)outputEvent)
        .eInverseRemove(this,
         DevsPackage.OUTPUT_EVENT__OUTPUT_FUNCTION,
          OutputEvent.class, msgs);
      return basicSetOutputEvent((OutputEvent)otherEnd,
       msgs);
    case DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL:
      if (eInternalContainer() != null)
        msgs = eBasicRemoveFromContainer(msgs);
        return basicSetAtomicModel((AtomicModel)otherEnd,
         msgs);
  }
  return super.eInverseAdd(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseRemove(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.OUTPUT_FUNCTION__SOURCE:
      return basicSetSource(null, msgs);
    case DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT:
      return basicSetOutputEvent(null, msgs);
    case DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL:
```

```
      return basicSetAtomicModel(null, msgs);
  }
  return super.eInverseRemove(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain
 eBasicRemoveFromContainerFeature(NotificationChain
  msgs) {
  switch (eContainerFeatureID()) {
    case DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL:
      return eInternalContainer().eInverseRemove(this,
       DevsPackage.ATOMIC_MODEL__OUTPUT_FUNCTION,
        AtomicModel.class, msgs);
  }
  return super.eBasicRemoveFromContainerFeature(msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public Object eGet(int featureID, boolean resolve,
 boolean coreType) {
  switch (featureID) {
    case DevsPackage.OUTPUT_FUNCTION__NAME:
      return getName();
    case DevsPackage.OUTPUT_FUNCTION__SOURCE:
      if (resolve) return getSource();
      return basicGetSource();
    case DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT:
      if (resolve) return getOutputEvent();
      return basicGetOutputEvent();
    case DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL:
      return getAtomicModel();
  }
  return super.eGet(featureID, resolve, coreType);
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
  */
@Override
public void eSet(int featureID, Object newValue) {
  switch (featureID) {
    case DevsPackage.OUTPUT_FUNCTION__NAME:
      setName((String)newValue);
      return;
    case DevsPackage.OUTPUT_FUNCTION__SOURCE:
      setSource((State)newValue);
      return;
    case DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT:
      setOutputEvent((OutputEvent)newValue);
      return;
    case DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL:
      setAtomicModel((AtomicModel)newValue);
      return;
  }
  super.eSet(featureID, newValue);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public void eUnset(int featureID) {
  switch (featureID) {
    case DevsPackage.OUTPUT_FUNCTION__NAME:
      setName(NAME_EDEFAULT);
      return;
    case DevsPackage.OUTPUT_FUNCTION__SOURCE:
      setSource((State)null);
      return;
    case DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT:
      setOutputEvent((OutputEvent)null);
      return;
    case DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL:
      setAtomicModel((AtomicModel)null);
      return;
  }
  super.eUnset(featureID);
```

```
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public boolean eIsSet(int featureID) {
    switch (featureID) {
      case DevsPackage.OUTPUT_FUNCTION__NAME:
        return NAME_EDEFAULT == null ? name != null :
         !NAME_EDEFAULT.equals(name);
      case DevsPackage.OUTPUT_FUNCTION__SOURCE:
        return source != null;
      case DevsPackage.OUTPUT_FUNCTION__OUTPUT_EVENT:
        return outputEvent != null;
      case DevsPackage.OUTPUT_FUNCTION__ATOMIC_MODEL:
        return getAtomicModel() != null;
    }
    return super.eIsSet(featureID);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public String toString() {
    if (eIsProxy()) return super.toString();

    StringBuffer result = new
     StringBuffer(super.toString());
    result.append(" (name: ");
    result.append(name);
    result.append(')');
    return result.toString();
  }

} //OutputFunctionImpl
```

Implementación StateImpl.java.

```
/**
```

```
*/
package devs.impl;

import devs.AtomicModel;
import devs.DevsPackage;
import devs.OutputFunction;
import devs.State;
import devs.Transition;

import java.util.Collection;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.notify.NotificationChain;

import org.eclipse.emf.common.util.EList;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.EObjectImpl;

import org.eclipse.emf.ecore.util.EObjectWithInverseResolvingEList;
import org.eclipse.emf.ecore.util.EcoreUtil;
import org.eclipse.emf.ecore.util.InternalEList;

/**
* <!-- begin-user-doc -->
* An implementation of the model object
* '<em><b>State</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link devs.impl.StateImpl#getLifeTime <em>Life
* Time</em>}</li>
*   <li>{@link devs.impl.StateImpl#getName
* <em>Name</em>}</li>
*   <li>{@link devs.impl.StateImpl#getIn <em>In</em>}</li>
*   <li>{@link devs.impl.StateImpl#getOut
* <em>Out</em>}</li>
*   <li>{@link devs.impl.StateImpl#getOutF <em>Out
* F</em>}</li>
*   <li>{@link devs.impl.StateImpl#getAtomicModel
* <em>Atomic Model</em>}</li>
* </ul>
```

```
* </p>
*
* @generated
*/
public class StateImpl extends EObjectImpl implements State {
  /**
  * The default value of the '{@link #getLifeTime()
  * <em>Life Time</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getLifeTime()
    * @generated
  * @ordered
  */
  protected static final double LIFE_TIME_EDEFAULT = 0.0;

  /**
  * The cached value of the '{@link #getLifeTime()
  * <em>Life Time</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getLifeTime()
  * @generated
  * @ordered
  */
  protected double lifeTime = LIFE_TIME_EDEFAULT;

  /**
  * The default value of the '{@link #getName()
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
    * <!-- end-user-doc -->
    * @see #getName()
  * @generated
  * @ordered
  */
  protected static final String NAME_EDEFAULT = null;

  /**
  * The cached value of the '{@link #getName()
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
    * @ordered
```

```
*/
protected String name = NAME_EDEFAULT;

/**
* The cached value of the '{@link #getIn() <em>In</em>}'
* reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getIn()
* @generated
* @ordered
*/
protected EList<Transition> in;

/**
* The cached value of the '{@link #getOut()
* <em>Out</em>}' reference list.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getOut()
* @generated
* @ordered
*/
protected EList<Transition> out;

/**
* The cached value of the '{@link #getOutF() <em>Out
* F</em>}' reference.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #getOutF()
* @generated
* @ordered
*/
protected OutputFunction outF;

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
protected StateImpl() {
  super();
}

/**
```

```
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
protected EClass eStaticClass() {
  return DevsPackage.Literals.STATE;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public double getLifeTime() {
  return lifeTime;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setLifeTime(double newLifeTime) {
  double oldLifeTime = lifeTime;
  lifeTime = newLifeTime;
  if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET, DevsPackage.STATE__LIFE_TIME,
      oldLifeTime, lifeTime));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public String getName() {
  return name;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
```

```
public void setName(String newName) {
  String oldName = name;
  name = newName;
  if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET, DevsPackage.STATE__NAME, oldName,
      name));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EList<Transition> getIn() {
  if (in == null) {
    in = new EObjectWithInverseResolvingEList<Transition>(
     Transition.class, this, DevsPackage.STATE__IN,
      DevsPackage.TRANSITION__TARGET);
  }
  return in;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EList<Transition> getOut() {
  if (out == null) {
    out = new EObjectWithInverseResolvingEList<Transition>(
     Transition.class, this, DevsPackage.STATE__OUT,
      DevsPackage.TRANSITION__SOURCE);
  }
  return out;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public OutputFunction getOutF() {
  if (outF != null && outF.eIsProxy()) {
    InternalEObject oldOutF = (InternalEObject)outF;
    outF = (OutputFunction)eResolveProxy(oldOutF);
```

```
    if (outF != oldOutF) {
      if (eNotificationRequired())
        eNotify(new ENotificationImpl(this,
         Notification.RESOLVE, DevsPackage.STATE__OUT_F,
          oldOutF, outF));
    }
  }
  return outF;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public OutputFunction basicGetOutF() {
  return outF;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetOutF(OutputFunction
 newOutF, NotificationChain msgs) {
  OutputFunction oldOutF = outF;
  outF = newOutF;
  if (eNotificationRequired()) {
    ENotificationImpl notification = new ENotificationImpl(
     this, Notification.SET, DevsPackage.STATE__OUT_F,
      oldOutF, newOutF);
    if (msgs == null) msgs = notification;
     else msgs.add(notification);
  }
  return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setOutF(OutputFunction newOutF) {
  if (newOutF != outF) {
    NotificationChain msgs = null;
```

```
    if (outF != null)
      msgs = ((InternalEObject)outF)
      .eInverseRemove(this,
       DevsPackage.OUTPUT_FUNCTION__SOURCE,
        OutputFunction.class, msgs);
    if (newOutF != null)
      msgs = ((InternalEObject)newOutF)
      .eInverseAdd(this,
       DevsPackage.OUTPUT_FUNCTION__SOURCE,
        OutputFunction.class, msgs);
    msgs = basicSetOutF(newOutF, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET, DevsPackage.STATE__OUT_F,
      newOutF, newOutF));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public AtomicModel getAtomicModel() {
  if (eContainerFeatureID() !=
   DevsPackage.STATE__ATOMIC_MODEL) return null;
  return (AtomicModel)eInternalContainer();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetAtomicModel(AtomicModel
 newAtomicModel, NotificationChain msgs) {
  msgs = eBasicSetContainer((InternalEObject)newAtomicModel,
   DevsPackage.STATE__ATOMIC_MODEL, msgs);
  return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
```

```
*/
public void setAtomicModel(AtomicModel newAtomicModel) {
  if (newAtomicModel != eInternalContainer() ||
   (eContainerFeatureID() != DevsPackage.STATE__ATOMIC_MODEL && newAtomicModel !=
    null)) {
    if (EcoreUtil.isAncestor(this, newAtomicModel))
      throw new IllegalArgumentException("Recursive
       containment not allowed for " + toString());
    NotificationChain msgs = null;
    if (eInternalContainer() != null)
      msgs = eBasicRemoveFromContainer(msgs);
    if (newAtomicModel != null)
      msgs = ((InternalEObject)newAtomicModel)
      .eInverseAdd(this,
       DevsPackage.ATOMIC_MODEL__STATE,
        AtomicModel.class, msgs);
    msgs = basicSetAtomicModel(newAtomicModel, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET, DevsPackage.STATE__ATOMIC_MODEL,
      newAtomicModel, newAtomicModel));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@SuppressWarnings("unchecked")
@Override
public NotificationChain eInverseAdd(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.STATE__IN:
      return ((InternalEList<InternalEObject>)(InternalEList<?>)
       getIn()).basicAdd(otherEnd, msgs);
    case DevsPackage.STATE__OUT:
      return ((InternalEList<InternalEObject>)(InternalEList<?>)
       getOut()).basicAdd(otherEnd, msgs);
    case DevsPackage.STATE__OUT_F:
      if (outF != null)
        msgs = ((InternalEObject)outF)
        .eInverseRemove(this,
         DevsPackage.OUTPUT_FUNCTION__SOURCE,
```

```
          OutputFunction.class, msgs);
      return basicSetOutF((OutputFunction)otherEnd,
       msgs);
    case DevsPackage.STATE__ATOMIC_MODEL:
      if (eInternalContainer() != null)
        msgs = eBasicRemoveFromContainer(msgs);
      return basicSetAtomicModel((AtomicModel)otherEnd,
       msgs);
  }
  return super.eInverseAdd(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseRemove(InternalEObject otherEnd,
 int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.STATE__IN:
      return ((InternalEList<?>)getIn())
       .basicRemove(otherEnd, msgs);
    case DevsPackage.STATE__OUT:
      return ((InternalEList<?>)getOut())
       .basicRemove(otherEnd, msgs);
    case DevsPackage.STATE__OUT_F:
      return basicSetOutF(null, msgs);
    case DevsPackage.STATE__ATOMIC_MODEL:
      return basicSetAtomicModel(null, msgs);
  }
  return super.eInverseRemove(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eBasicRemoveFromContainerFeature(
 NotificationChain msgs) {
  switch (eContainerFeatureID()) {
    case DevsPackage.STATE__ATOMIC_MODEL:
      return eInternalContainer().eInverseRemove(this,
```

```
        DevsPackage.ATOMIC_MODEL__STATE,
         AtomicModel.class, msgs);
    }
    return super.eBasicRemoveFromContainerFeature(msgs);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @Override
  public Object eGet(int featureID, boolean resolve,
   boolean coreType) {
    switch (featureID) {
      case DevsPackage.STATE__LIFE_TIME:
        return getLifeTime();
      case DevsPackage.STATE__NAME:
        return getName();
      case DevsPackage.STATE__IN:
        return getIn();
      case DevsPackage.STATE__OUT:
        return getOut();
      case DevsPackage.STATE__OUT_F:
        if (resolve) return getOutF();
        return basicGetOutF();
      case DevsPackage.STATE__ATOMIC_MODEL:
        return getAtomicModel();
    }
    return super.eGet(featureID, resolve, coreType);
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  @SuppressWarnings("unchecked")
  @Override
  public void eSet(int featureID, Object newValue) {
    switch (featureID) {
      case DevsPackage.STATE__LIFE_TIME:
        setLifeTime((Double)newValue);
        return;
      case DevsPackage.STATE__NAME:
        setName((String)newValue);
```

```
      return;
    case DevsPackage.STATE__IN:
      getIn().clear();
      getIn().addAll((Collection<? extends
       Transition>)newValue);
      return;
    case DevsPackage.STATE__OUT:
      getOut().clear();
      getOut().addAll((Collection<? extends
       Transition>)newValue);
      return;
    case DevsPackage.STATE__OUT_F:
      setOutF((OutputFunction)newValue);
      return;
    case DevsPackage.STATE__ATOMIC_MODEL:
      setAtomicModel((AtomicModel)newValue);
      return;
  }
  super.eSet(featureID, newValue);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public void eUnset(int featureID) {
  switch (featureID) {
    case DevsPackage.STATE__LIFE_TIME:
      setLifeTime(LIFE_TIME_EDEFAULT);
      return;
    case DevsPackage.STATE__NAME:
      setName(NAME_EDEFAULT);
      return;
    case DevsPackage.STATE__IN:
      getIn().clear();
      return;
    case DevsPackage.STATE__OUT:
      getOut().clear();
      return;
    case DevsPackage.STATE__OUT_F:
      setOutF((OutputFunction)null);
      return;
    case DevsPackage.STATE__ATOMIC_MODEL:
      setAtomicModel((AtomicModel)null);
```

```
      return;
  }
  super.eUnset(featureID);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public boolean eIsSet(int featureID) {
  switch (featureID) {
    case DevsPackage.STATE__LIFE_TIME:
      return lifeTime != LIFE_TIME_EDEFAULT;
    case DevsPackage.STATE__NAME:
      return NAME_EDEFAULT == null ? name != null :
       !NAME_EDEFAULT.equals(name);
    case DevsPackage.STATE__IN:
      return in != null && !in.isEmpty();
    case DevsPackage.STATE__OUT:
      return out != null && !out.isEmpty();
    case DevsPackage.STATE__OUT_F:
      return outF != null;
    case DevsPackage.STATE__ATOMIC_MODEL:
      return getAtomicModel() != null;
  }
  return super.eIsSet(featureID);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public String toString() {
  if (eIsProxy()) return super.toString();

  StringBuffer result =
   new StringBuffer(super.toString());
  result.append(" (lifeTime: ");
  result.append(lifeTime);
  result.append(", name: ");
  result.append(name);
  result.append(')');
```

```
    return result.toString();
  }

} //StateImpl
```

Implementación TransitionImpl.java.

```
/**
*/
package devs.impl;

import devs.AtomicModel;
import devs.DevsPackage;
import devs.State;
import devs.Transition;

import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.notify.NotificationChain;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.InternalEObject;

import org.eclipse.emf.ecore.impl.ENotificationImpl;
import org.eclipse.emf.ecore.impl.EObjectImpl;

import org.eclipse.emf.ecore.util.EcoreUtil;

/**
* <!-- begin-user-doc -->
* An implementation of the model object
* '<em><b>Transition</b></em>'.
* <!-- end-user-doc -->
* <p>
* The following features are implemented:
* <ul>
*   <li>{@link devs.impl.TransitionImpl#getName
* <em>Name</em>}</li>
*   <li>{@link devs.impl.TransitionImpl#getSource
* <em>Source</em>}</li>
*   <li>{@link devs.impl.TransitionImpl#getTarget
* <em>Target</em>}</li>
*   <li>{@link devs.impl.TransitionImpl#getAtomicModel
* <em>Atomic Model</em>}</li>
* </ul>
* </p>
```

```
*
* @generated
*/
public class TransitionImpl extends
 EObjectImpl implements Transition {
  /**
  * The default value of the '{@link #getName()
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
  * @ordered
  */
  protected static final String NAME_EDEFAULT = null;

     /**
  * The cached value of the '{@link #getName()
  * <em>Name</em>}' attribute.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getName()
  * @generated
  * @ordered
  */
  protected String name = NAME_EDEFAULT;

  /**
  * The cached value of the '{@link #getSource()
  * <em>Source</em>}' reference.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getSource()
  * @generated
  * @ordered
  */
  protected State source;

  /**
  * The cached value of the '{@link #getTarget()
  * <em>Target</em>}' reference.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @see #getTarget()
  * @generated
    * @ordered
```

```
*/
protected State target;

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
protected TransitionImpl() {
  super();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
  */
@Override
protected EClass eStaticClass() {
  return DevsPackage.Literals.TRANSITION;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public String getName() {
  return name;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setName(String newName) {
  String oldName = name;
  name = newName;
  if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET, DevsPackage.TRANSITION__NAME,
      oldName, name));
}

/**
```

```
* <!-- begin-user-doc -->
  * <!-- end-user-doc -->
* @generated
*/
public State getSource() {
  if (source != null && source.eIsProxy()) {
    InternalEObject oldSource = (InternalEObject)source;
    source = (State)eResolveProxy(oldSource);
    if (source != oldSource) {
      if (eNotificationRequired())
        eNotify(new ENotificationImpl(this,
         Notification.RESOLVE,
          DevsPackage.TRANSITION__SOURCE, oldSource,
           source));
    }
  }
  return source;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public State basicGetSource() {
  return source;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetSource(State newSource,
 NotificationChain msgs) {
  State oldSource = source;
  source = newSource;
  if (eNotificationRequired()) {
    ENotificationImpl notification = new ENotificationImpl(
     this, Notification.SET,
      DevsPackage.TRANSITION__SOURCE, oldSource,
       newSource);
    if (msgs == null) msgs = notification;
     else msgs.add(notification);
  }
  return msgs;
```

```
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setSource(State newSource) {
  if (newSource != source) {
    NotificationChain msgs = null;
    if (source != null)
      msgs = ((InternalEObject)source)
       .eInverseRemove(this, DevsPackage.STATE__OUT,
        State.class, msgs);
    if (newSource != null)
      msgs = ((InternalEObject)newSource)
       .eInverseAdd(this, DevsPackage.STATE__OUT,
        State.class, msgs);
    msgs = basicSetSource(newSource, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET, DevsPackage.TRANSITION__SOURCE,
      newSource, newSource));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public State getTarget() {
  if (target != null && target.eIsProxy()) {
    InternalEObject oldTarget = (InternalEObject)target;
    target = (State)eResolveProxy(oldTarget);
    if (target != oldTarget) {
      if (eNotificationRequired())
        eNotify(new ENotificationImpl(this,
         Notification.RESOLVE,
          DevsPackage.TRANSITION__TARGET, oldTarget,
           target));
    }
  }
  return target;
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public State basicGetTarget() {
  return target;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetTarget(State newTarget,
 NotificationChain msgs) {
  State oldTarget = target;
  target = newTarget;
  if (eNotificationRequired()) {
    ENotificationImpl notification = new ENotificationImpl(
     this, Notification.SET,
      DevsPackage.TRANSITION__TARGET, oldTarget,
       newTarget);
    if (msgs == null) msgs = notification;
     else msgs.add(notification);
  }
  return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setTarget(State newTarget) {
  if (newTarget != target) {
    NotificationChain msgs = null;
    if (target != null)
      msgs = ((InternalEObject)target)
      .eInverseRemove(this, DevsPackage.STATE__IN,
       State.class, msgs);
    if (newTarget != null)
      msgs = ((InternalEObject)newTarget)
      .eInverseAdd(this, DevsPackage.STATE__IN,
       State.class, msgs);
```

```
    msgs = basicSetTarget(newTarget, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET, DevsPackage.TRANSITION__TARGET,
      newTarget, newTarget));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public AtomicModel getAtomicModel() {
  if (eContainerFeatureID() !=
   DevsPackage.TRANSITION__ATOMIC_MODEL) return null;
  return (AtomicModel)eInternalContainer();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public NotificationChain basicSetAtomicModel(AtomicModel
 newAtomicModel, NotificationChain msgs) {
  msgs = eBasicSetContainer((InternalEObject)newAtomicModel,
   DevsPackage.TRANSITION__ATOMIC_MODEL, msgs);
  return msgs;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void setAtomicModel(AtomicModel newAtomicModel) {
  if (newAtomicModel != eInternalContainer() ||
   (eContainerFeatureID() !=
    DevsPackage.TRANSITION__ATOMIC_MODEL &&
     newAtomicModel != null)) {
      if (EcoreUtil.isAncestor(this, newAtomicModel))
        throw new IllegalArgumentException("Recursive
         containment not allowed for " + toString());
      NotificationChain msgs = null;
```

```
      if (eInternalContainer() != null)
        msgs = eBasicRemoveFromContainer(msgs);
      if (newAtomicModel != null)
        msgs = ((InternalEObject)newAtomicModel)
         .eInverseAdd(this,
          DevsPackage.ATOMIC_MODEL__TRANSITION,
           AtomicModel.class, msgs);
      msgs = basicSetAtomicModel(newAtomicModel, msgs);
      if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
    eNotify(new ENotificationImpl(this,
     Notification.SET,
      DevsPackage.TRANSITION__ATOMIC_MODEL,
       newAtomicModel, newAtomicModel));
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseAdd(InternalEObject otherEnd,
 int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.TRANSITION__SOURCE:
      if (source != null)
        msgs = ((InternalEObject)source)
         .eInverseRemove(this, DevsPackage.STATE__OUT,
          State.class, msgs);
        return basicSetSource((State)otherEnd, msgs);
    case DevsPackage.TRANSITION__TARGET:
      if (target != null)
        msgs = ((InternalEObject)target)
         .eInverseRemove(this, DevsPackage.STATE__IN,
          State.class, msgs);
      return basicSetTarget((State)otherEnd, msgs);
    case DevsPackage.TRANSITION__ATOMIC_MODEL:
      if (eInternalContainer() != null)
        msgs = eBasicRemoveFromContainer(msgs);
        return basicSetAtomicModel((AtomicModel)otherEnd,
         msgs);
  }
  return super.eInverseAdd(otherEnd, featureID, msgs);
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eInverseRemove(InternalEObject
 otherEnd, int featureID, NotificationChain msgs) {
  switch (featureID) {
    case DevsPackage.TRANSITION__SOURCE:
      return basicSetSource(null, msgs);
    case DevsPackage.TRANSITION__TARGET:
      return basicSetTarget(null, msgs);
    case DevsPackage.TRANSITION__ATOMIC_MODEL:
      return basicSetAtomicModel(null, msgs);
  }
  return super.eInverseRemove(otherEnd, featureID, msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public NotificationChain eBasicRemoveFromContainerFeature(
 NotificationChain msgs) {
  switch (eContainerFeatureID()) {
    case DevsPackage.TRANSITION__ATOMIC_MODEL:
      return eInternalContainer().eInverseRemove(this,
       DevsPackage.ATOMIC_MODEL__TRANSITION,
        AtomicModel.class, msgs);
  }
  return super.eBasicRemoveFromContainerFeature(msgs);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public Object eGet(int featureID, boolean resolve,
 boolean coreType) {
  switch (featureID) {
```

```
    case DevsPackage.TRANSITION__NAME:
      return getName();
    case DevsPackage.TRANSITION__SOURCE:
      if (resolve) return getSource();
        return basicGetSource();
    case DevsPackage.TRANSITION__TARGET:
      if (resolve) return getTarget();
      return basicGetTarget();
    case DevsPackage.TRANSITION__ATOMIC_MODEL:
      return getAtomicModel();
  }
  return super.eGet(featureID, resolve, coreType);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public void eSet(int featureID, Object newValue) {
  switch (featureID) {
    case DevsPackage.TRANSITION__NAME:
      setName((String)newValue);
         return;
    case DevsPackage.TRANSITION__SOURCE:
      setSource((State)newValue);
      return;
    case DevsPackage.TRANSITION__TARGET:
      setTarget((State)newValue);
      return;
    case DevsPackage.TRANSITION__ATOMIC_MODEL:
      setAtomicModel((AtomicModel)newValue);
      return;
  }
  super.eSet(featureID, newValue);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public void eUnset(int featureID) {
  switch (featureID) {
```

```
    case DevsPackage.TRANSITION__NAME:
      setName(NAME_EDEFAULT);
      return;
    case DevsPackage.TRANSITION__SOURCE:
      setSource((State)null);
      return;
    case DevsPackage.TRANSITION__TARGET:
      setTarget((State)null);
      return;
    case DevsPackage.TRANSITION__ATOMIC_MODEL:
      setAtomicModel((AtomicModel)null);
      return;
  }
  super.eUnset(featureID);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public boolean eIsSet(int featureID) {
  switch (featureID) {
    case DevsPackage.TRANSITION__NAME:
      return NAME_EDEFAULT == null ? name != null :
       !NAME_EDEFAULT.equals(name);
    case DevsPackage.TRANSITION__SOURCE:
      return source != null;
    case DevsPackage.TRANSITION__TARGET:
      return target != null;
    case DevsPackage.TRANSITION__ATOMIC_MODEL:
      return getAtomicModel() != null;
  }
  return super.eIsSet(featureID);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public String toString() {
  if (eIsProxy()) return super.toString();
```

```
    StringBuffer result = new
     StringBuffer(super.toString());
    result.append(" (name: ");
    result.append(name);
    result.append(')');
    return result.toString();
  }

} //TransitionImpl
```

Implementación DevsFactoryImpl.java.

```
/**
*/
package devs.impl;

import devs.*;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.EPackage;

import org.eclipse.emf.ecore.impl.EFactoryImpl;

import org.eclipse.emf.ecore.plugin.EcorePlugin;

/**
* <!-- begin-user-doc -->
* An implementation of the model <b>Factory</b>.
* <!-- end-user-doc -->
* @generated
*/
public class DevsFactoryImpl extends
 EFactoryImpl implements DevsFactory {
  /**
  * Creates the default factory implementation.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  public static DevsFactory init() {
    try {
      DevsFactory theDevsFactory =
       (DevsFactory)EPackage.Registry.INSTANCE.getEFactory(
        DevsPackage.eNS_URI);
```

```
    if (theDevsFactory != null) {
      return theDevsFactory;
    }
  }
  catch (Exception exception) {
    EcorePlugin.INSTANCE.log(exception);
  }
  return new DevsFactoryImpl();
}

/**
* Creates an instance of the factory.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public DevsFactoryImpl() {
  super();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
@Override
public EObject create(EClass eClass) {
  switch (eClass.getClassifierID()) {
    case DevsPackage.ATOMIC_MODEL:
     return createAtomicModel();
    case DevsPackage.STATE:
     return createState();
    case DevsPackage.EVENT:
     return createEvent();
      case DevsPackage.INPUT_EVENT:
       return createInputEvent();
    case DevsPackage.OUTPUT_EVENT:
     return createOutputEvent();
    case DevsPackage.TRANSITION:
     return createTransition();
    case DevsPackage.INTERNAL_TRANSITION:
     return createInternalTransition();
    case DevsPackage.EXTERNAL_TRANSITION:
     return createExternalTransition();
    case DevsPackage.OUTPUT_FUNCTION:
     return createOutputFunction();
```

```
    default:
      throw new IllegalArgumentException("The class '" +
       eClass.getName() + "' is not a valid classifier");
  }
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public AtomicModel createAtomicModel() {
  AtomicModelImpl atomicModel = new AtomicModelImpl();
  return atomicModel;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public State createState() {
  StateImpl state = new StateImpl();
  return state;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public Event createEvent() {
  EventImpl event = new EventImpl();
  return event;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public InputEvent createInputEvent() {
  InputEventImpl inputEvent = new InputEventImpl();
  return inputEvent;
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public OutputEvent createOutputEvent() {
  OutputEventImpl outputEvent = new OutputEventImpl();
  return outputEvent;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public Transition createTransition() {
  TransitionImpl transition = new TransitionImpl();
  return transition;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public InternalTransition createInternalTransition() {
  InternalTransitionImpl internalTransition =
   new InternalTransitionImpl();
  return internalTransition;
}

/**
   * <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public ExternalTransition createExternalTransition() {
  ExternalTransitionImpl externalTransition =
   new ExternalTransitionImpl();
  return externalTransition;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
```

```
  */
  public OutputFunction createOutputFunction() {
    OutputFunctionImpl outputFunction =
     new OutputFunctionImpl();
    return outputFunction;
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  public DevsPackage getDevsPackage() {
    return (DevsPackage)getEPackage();
  }

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @deprecated
  * @generated
  */
  @Deprecated
  public static DevsPackage getPackage() {
    return DevsPackage.eINSTANCE;
  }

} //DevsFactoryImpl
```

Implementación DevsPackageImpl.java.

```
/**
*/
package devs.impl;

import devs.AtomicModel;
import devs.DevsFactory;
import devs.DevsPackage;
import devs.Event;
import devs.ExternalTransition;
import devs.InputEvent;
import devs.InternalTransition;
import devs.OutputEvent;
import devs.OutputFunction;
import devs.State;
```

```
import devs.Transition;

import org.eclipse.emf.ecore.EAttribute;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EPackage;
import org.eclipse.emf.ecore.EReference;

import org.eclipse.emf.ecore.impl.EPackageImpl;

/**
* <!-- begin-user-doc -->
* An implementation of the model <b>Package</b>.
* <!-- end-user-doc -->
* @generated
*/
public class DevsPackageImpl extends
 EPackageImpl implements DevsPackage {
  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  private EClass atomicModelEClass = null;

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  private EClass stateEClass = null;

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  private EClass eventEClass = null;

  /**
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  private EClass inputEventEClass = null;

  /**
```

```
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
private EClass outputEventEClass = null;

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
private EClass transitionEClass = null;

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
private EClass internalTransitionEClass = null;

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
private EClass externalTransitionEClass = null;

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
private EClass outputFunctionEClass = null;

/**
* Creates an instance of the model <b>Package</b>,
* registered with
* {@link org.eclipse.emf.ecore.EPackage.Registry
* EPackage.Registry} by the package
* package URI value.
* <p>Note: the correct way to create the package is via
* the static
* factory method {@link #init init()}, which also
* performs
* initialization of the package, or returns the
* registered package,
```

```
* if one already exists.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see org.eclipse.emf.ecore.EPackage.Registry
* @see devs.DevsPackage#eNS_URI
* @see #init()
* @generated
*/
private DevsPackageImpl() {
  super(eNS_URI, DevsFactory.eINSTANCE);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
private static boolean isInited = false;

/**
* Creates, registers, and initializes the <b>Package</b>
* for this model, and for any others upon which it
* depends.
*
* <p>This method is used to initialize {@link
* DevsPackage#eINSTANCE} when that field is accessed.
* Clients should not invoke it directly. Instead, they
* should simply access that field to obtain the package.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @see #eNS_URI
* @see #createPackageContents()
* @see #initializePackageContents()
* @generated
*/
public static DevsPackage init() {
  if (isInited) return
   (DevsPackage)EPackage.Registry.INSTANCE
    .getEPackage(DevsPackage.eNS_URI);

// Obtain or create and register package
DevsPackageImpl theDevsPackage =
 (DevsPackageImpl)(EPackage.Registry.INSTANCE.get(eNS_URI) instanceof DevsPackageI
  EPackage.Registry.INSTANCE.get(eNS_URI) :
   new DevsPackageImpl());
```

```
isInited = true;

// Create package meta-data objects
theDevsPackage.createPackageContents();

// Initialize created meta-data
theDevsPackage.initializePackageContents();

// Mark meta-data to indicate it can't be changed
theDevsPackage.freeze();


  // Update the registry and return the package
  EPackage.Registry.INSTANCE.put(DevsPackage.eNS_URI,
   theDevsPackage);
  return theDevsPackage;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EClass getAtomicModel() {
  return atomicModelEClass;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EAttribute getAtomicModel_Name() {
  return (EAttribute)atomicModelEClass
   .getEStructuralFeatures().get(0);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getAtomicModel_State() {
  return (EReference)atomicModelEClass
   .getEStructuralFeatures().get(1);
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getAtomicModel_Event() {
  return (EReference)atomicModelEClass
   .getEStructuralFeatures().get(2);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getAtomicModel_Transition() {
  return (EReference)atomicModelEClass
   .getEStructuralFeatures().get(3);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getAtomicModel_OutputFunction() {
  return (EReference)atomicModelEClass
   .getEStructuralFeatures().get(4);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EClass getState() {
  return stateEClass;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EAttribute getState_LifeTime() {
```

```
  return (EAttribute)stateEClass
   .getEStructuralFeatures().get(0);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EAttribute getState_Name() {
  return (EAttribute)stateEClass
   .getEStructuralFeatures().get(1);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getState_In() {
  return (EReference)stateEClass
   .getEStructuralFeatures().get(2);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getState_Out() {
  return (EReference)stateEClass
   .getEStructuralFeatures().get(3);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getState_OutF() {
  return (EReference)stateEClass
   .getEStructuralFeatures().get(4);
}

/**
* <!-- begin-user-doc -->
```

```
* <!-- end-user-doc -->
* @generated
*/
public EReference getState_AtomicModel() {
  return (EReference)stateEClass
   .getEStructuralFeatures().get(5);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EClass getEvent() {
  return eventEClass;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EAttribute getEvent_Name() {
  return (EAttribute)eventEClass
   .getEStructuralFeatures().get(0);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getEvent_AtomicModel() {
  return (EReference)eventEClass
   .getEStructuralFeatures().get(1);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EClass getInputEvent() {
  return inputEventEClass;
}
```

```
/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getInputEvent_ExternalTransition() {
  return (EReference)inputEventEClass
   .getEStructuralFeatures().get(0);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EClass getOutputEvent() {
  return outputEventEClass;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getOutputEvent_OutputFunction() {
  return (EReference)outputEventEClass
   .getEStructuralFeatures().get(0);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EClass getTransition() {
  return transitionEClass;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EAttribute getTransition_Name() {
  return (EAttribute)transitionEClass
   .getEStructuralFeatures().get(0);
```

```
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getTransition_Source() {
  return (EReference)transitionEClass
   .getEStructuralFeatures().get(1);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getTransition_Target() {
  return (EReference)transitionEClass
   .getEStructuralFeatures().get(2);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getTransition_AtomicModel() {
  return (EReference)transitionEClass
   .getEStructuralFeatures().get(3);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EClass getInternalTransition() {
  return internalTransitionEClass;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
```

```
public EClass getExternalTransition() {
  return externalTransitionEClass;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getExternalTransition_InputEvent() {
  return (EReference)externalTransitionEClass
   .getEStructuralFeatures().get(0);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EClass getOutputFunction() {
  return outputFunctionEClass;
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EAttribute getOutputFunction_Name() {
  return (EAttribute)outputFunctionEClass
   .getEStructuralFeatures().get(0);
  }

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getOutputFunction_Source() {
  return (EReference)outputFunctionEClass
   .getEStructuralFeatures().get(1);
}

/**
  * <!-- begin-user-doc -->
* <!-- end-user-doc -->
```

```
* @generated
*/
public EReference getOutputFunction_OutputEvent() {
  return (EReference)outputFunctionEClass
   .getEStructuralFeatures().get(2);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public EReference getOutputFunction_AtomicModel() {
  return (EReference)outputFunctionEClass
   .getEStructuralFeatures().get(3);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public DevsFactory getDevsFactory() {
  return (DevsFactory)getEFactoryInstance();
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
private boolean isCreated = false;

/**
* Creates the meta-model objects for the package.
* This method is
* guarded to have no affect on any invocation but its
* first.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void createPackageContents() {
  if (isCreated) return;
    isCreated = true;
```

```
// Create classes and their features
atomicModelEClass = createEClass(ATOMIC_MODEL);
createEAttribute(atomicModelEClass,
 ATOMIC_MODEL__NAME);
createEReference(atomicModelEClass,
 ATOMIC_MODEL__STATE);
createEReference(atomicModelEClass,
 ATOMIC_MODEL__EVENT);
createEReference(atomicModelEClass,
 ATOMIC_MODEL__TRANSITION);
createEReference(atomicModelEClass,
 ATOMIC_MODEL__OUTPUT_FUNCTION);

stateEClass = createEClass(STATE);
createEAttribute(stateEClass, STATE__LIFE_TIME);
createEAttribute(stateEClass, STATE__NAME);
createEReference(stateEClass, STATE__IN);
createEReference(stateEClass, STATE__OUT);
createEReference(stateEClass, STATE__OUT_F);
createEReference(stateEClass, STATE__ATOMIC_MODEL);

eventEClass = createEClass(EVENT);
createEAttribute(eventEClass, EVENT__NAME);
createEReference(eventEClass, EVENT__ATOMIC_MODEL);

inputEventEClass = createEClass(INPUT_EVENT);
createEReference(inputEventEClass,
 INPUT_EVENT__EXTERNAL_TRANSITION);

outputEventEClass = createEClass(OUTPUT_EVENT);
createEReference(outputEventEClass,
 OUTPUT_EVENT__OUTPUT_FUNCTION);

transitionEClass = createEClass(TRANSITION);
createEAttribute(transitionEClass, TRANSITION__NAME);
createEReference(transitionEClass, TRANSITION__SOURCE);
createEReference(transitionEClass, TRANSITION__TARGET);
createEReference(transitionEClass,
 TRANSITION__ATOMIC_MODEL);

internalTransitionEClass =
 createEClass(INTERNAL_TRANSITION);

externalTransitionEClass =
 createEClass(EXTERNAL_TRANSITION);
createEReference(externalTransitionEClass,
```

```
    EXTERNAL_TRANSITION__INPUT_EVENT);

  outputFunctionEClass = createEClass(OUTPUT_FUNCTION);
  createEAttribute(outputFunctionEClass,
   OUTPUT_FUNCTION__NAME);
  createEReference(outputFunctionEClass,
   OUTPUT_FUNCTION__SOURCE);
  createEReference(outputFunctionEClass,
   OUTPUT_FUNCTION__OUTPUT_EVENT);
  createEReference(outputFunctionEClass,
   OUTPUT_FUNCTION__ATOMIC_MODEL);
}

/**
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
private boolean isInitialized = false;

/**
* Complete the initialization of the package and its
* meta-model.  This
* method is guarded to have no affect on any invocation
* but its first.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public void initializePackageContents() {
  if (isInitialized) return;
  isInitialized = true;

  // Initialize package
  setName(eNAME);
  setNsPrefix(eNS_PREFIX);
  setNsURI(eNS_URI);

  // Create type parameters

  // Set bounds for type parameters

  // Add supertypes to classes
  inputEventEClass.getESuperTypes().add(this.getEvent());
  outputEventEClass.getESuperTypes().add(this.getEvent());
  internalTransitionEClass.getESuperTypes()
```

```
 .add(this.getTransition());
externalTransitionEClass.getESuperTypes()
 .add(this.getTransition());

// Initialize classes, features, and operations;
// add parameters
initEClass(atomicModelEClass, AtomicModel.class,
 "AtomicModel", !IS_ABSTRACT, !IS_INTERFACE,
  IS_GENERATED_INSTANCE_CLASS);
initEAttribute(getAtomicModel_Name(),
 ecorePackage.getEString(), "name", null, 0, 1,
  AtomicModel.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
    !IS_DERIVED, IS_ORDERED);
initEReference(getAtomicModel_State(),
 this.getState(), this.getState_AtomicModel(),
  "state", null, 1, -1, AtomicModel.class,
   !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
    IS_COMPOSITE, !IS_RESOLVE_PROXIES, !IS_UNSETTABLE,
     IS_UNIQUE, !IS_DERIVED, IS_ORDERED);
initEReference(getAtomicModel_Event(),
 this.getEvent(), this.getEvent_AtomicModel(),
  "event", null, 0, -1, AtomicModel.class,
   !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
    IS_COMPOSITE, !IS_RESOLVE_PROXIES, !IS_UNSETTABLE,
     IS_UNIQUE, !IS_DERIVED, IS_ORDERED);
initEReference(getAtomicModel_Transition(),
 this.getTransition(),
  this.getTransition_AtomicModel(), "transition",
   null, 0, -1, AtomicModel.class, !IS_TRANSIENT,
    !IS_VOLATILE, IS_CHANGEABLE, IS_COMPOSITE,
     !IS_RESOLVE_PROXIES, !IS_UNSETTABLE, IS_UNIQUE,
      !IS_DERIVED, IS_ORDERED);
initEReference(getAtomicModel_OutputFunction(),
 this.getOutputFunction(),
  this.getOutputFunction_AtomicModel(),
   "outputFunction", null, 0, -1, AtomicModel.class,
    !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
     IS_COMPOSITE, !IS_RESOLVE_PROXIES,
      !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
       IS_ORDERED);

initEClass(stateEClass, State.class, "State",
 !IS_ABSTRACT, !IS_INTERFACE,
  IS_GENERATED_INSTANCE_CLASS);
```

```
initEAttribute(getState_LifeTime(),
 ecorePackage.getEDouble(), "lifeTime", null, 0, 1,
  State.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
    !IS_DERIVED, IS_ORDERED);
initEAttribute(getState_Name(),
 ecorePackage.getEString(), "name", null, 0, 1,
  State.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
    !IS_DERIVED, IS_ORDERED);
initEReference(getState_In(), this.getTransition(),
 this.getTransition_Target(), "in", null, 0, -1,
  State.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_COMPOSITE, IS_RESOLVE_PROXIES,
    !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
     IS_ORDERED);
initEReference(getState_Out(), this.getTransition(),
 this.getTransition_Source(), "out", null, 0, -1,
  State.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_COMPOSITE, IS_RESOLVE_PROXIES,
    !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
     IS_ORDERED);
initEReference(getState_OutF(),
 this.getOutputFunction(),
  this.getOutputFunction_Source(), "outF", null, 1, 1,
   State.class, !IS_TRANSIENT, !IS_VOLATILE,
    IS_CHANGEABLE, !IS_COMPOSITE, IS_RESOLVE_PROXIES,
     !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
      IS_ORDERED);
initEReference(getState_AtomicModel(),
 this.getAtomicModel(), this.getAtomicModel_State(),
  "atomicModel", null, 1, 1, State.class,
   !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
    !IS_COMPOSITE, !IS_RESOLVE_PROXIES,
     !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
      IS_ORDERED);

initEClass(eventEClass, Event.class, "Event",
 !IS_ABSTRACT, !IS_INTERFACE,
  IS_GENERATED_INSTANCE_CLASS);
  initEAttribute(getEvent_Name(),
   ecorePackage.getEString(), "name", null, 0, 1,
    Event.class, !IS_TRANSIENT, !IS_VOLATILE,
     IS_CHANGEABLE, !IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
      !IS_DERIVED, IS_ORDERED);
initEReference(getEvent_AtomicModel(),
```

```
 this.getAtomicModel(), this.getAtomicModel_Event(),
  "atomicModel", null, 1, 1, Event.class,
   !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
    !IS_COMPOSITE, !IS_RESOLVE_PROXIES,
     !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
      IS_ORDERED);

initEClass(inputEventEClass, InputEvent.class,
 "InputEvent", !IS_ABSTRACT, !IS_INTERFACE,
  IS_GENERATED_INSTANCE_CLASS);
    initEReference(getInputEvent_ExternalTransition(),
     this.getExternalTransition(),
      this.getExternalTransition_InputEvent(),
       "externalTransition", null, 0, -1,
        InputEvent.class, !IS_TRANSIENT, !IS_VOLATILE,
         IS_CHANGEABLE, !IS_COMPOSITE, IS_RESOLVE_PROXIES,
          !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
           IS_ORDERED);

initEClass(outputEventEClass, OutputEvent.class,
 "OutputEvent", !IS_ABSTRACT, !IS_INTERFACE,
  IS_GENERATED_INSTANCE_CLASS);
initEReference(getOutputEvent_OutputFunction(),
 this.getOutputFunction(),
  this.getOutputFunction_OutputEvent(),
   "outputFunction", null, 0, -1, OutputEvent.class,
    !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
     !IS_COMPOSITE, IS_RESOLVE_PROXIES,
      !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
       IS_ORDERED);

initEClass(transitionEClass, Transition.class,
 "Transition", !IS_ABSTRACT, !IS_INTERFACE,
  IS_GENERATED_INSTANCE_CLASS);
initEAttribute(getTransition_Name(),
 ecorePackage.getEString(), "name", null, 0, 1,
  Transition.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
    !IS_DERIVED, IS_ORDERED);
initEReference(getTransition_Source(), this.getState(),
 this.getState_Out(), "source", null, 1, 1,
  Transition.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_COMPOSITE, IS_RESOLVE_PROXIES,
    !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED, IS_ORDERED);
initEReference(getTransition_Target(), this.getState(),
 this.getState_In(), "target", null, 1, 1,
```

```
  Transition.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_COMPOSITE, IS_RESOLVE_PROXIES,
    !IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED, IS_ORDERED);
initEReference(getTransition_AtomicModel(),
 this.getAtomicModel(),
  this.getAtomicModel_Transition(), "atomicModel",
   null, 1, 1, Transition.class, !IS_TRANSIENT,
    !IS_VOLATILE, IS_CHANGEABLE, !IS_COMPOSITE,
     !IS_RESOLVE_PROXIES, !IS_UNSETTABLE, IS_UNIQUE,
      !IS_DERIVED, IS_ORDERED);

initEClass(internalTransitionEClass,
 InternalTransition.class, "InternalTransition",
  !IS_ABSTRACT, !IS_INTERFACE,
   IS_GENERATED_INSTANCE_CLASS);

initEClass(externalTransitionEClass,
 ExternalTransition.class, "ExternalTransition",
  !IS_ABSTRACT, !IS_INTERFACE,
   IS_GENERATED_INSTANCE_CLASS);
initEReference(getExternalTransition_InputEvent(),
 this.getInputEvent(),
  this.getInputEvent_ExternalTransition(),
   "inputEvent", null, 1, 1, ExternalTransition.class,
    !IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
     !IS_COMPOSITE, IS_RESOLVE_PROXIES, !IS_UNSETTABLE,
      IS_UNIQUE, !IS_DERIVED, IS_ORDERED);

initEClass(outputFunctionEClass, OutputFunction.class,
 "OutputFunction", !IS_ABSTRACT, !IS_INTERFACE,
  IS_GENERATED_INSTANCE_CLASS);
initEAttribute(getOutputFunction_Name(),
 ecorePackage.getEString(), "name", null, 0, 1,
  OutputFunction.class, !IS_TRANSIENT, !IS_VOLATILE,
   IS_CHANGEABLE, !IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
    !IS_DERIVED, IS_ORDERED);
initEReference(getOutputFunction_Source(),
 this.getState(), this.getState_OutF(), "source", null,
  1, 1, OutputFunction.class, !IS_TRANSIENT,
   !IS_VOLATILE, IS_CHANGEABLE, !IS_COMPOSITE,
    IS_RESOLVE_PROXIES, !IS_UNSETTABLE, IS_UNIQUE,
     !IS_DERIVED, IS_ORDERED);
   initEReference(getOutputFunction_OutputEvent(),
    this.getOutputEvent(),
     this.getOutputEvent_OutputFunction(), "outputEvent",
      null, 1, 1, OutputFunction.class, !IS_TRANSIENT,
```

```
          !IS_VOLATILE, IS_CHANGEABLE, !IS_COMPOSITE,
           IS_RESOLVE_PROXIES, !IS_UNSETTABLE, IS_UNIQUE,
            !IS_DERIVED, IS_ORDERED);
    initEReference(getOutputFunction_AtomicModel(),
     this.getAtomicModel(),
      this.getAtomicModel_OutputFunction(), "atomicModel",
       null, 1, 1, OutputFunction.class, !IS_TRANSIENT,
        !IS_VOLATILE, IS_CHANGEABLE, !IS_COMPOSITE,
         !IS_RESOLVE_PROXIES, !IS_UNSETTABLE, IS_UNIQUE,
          !IS_DERIVED, IS_ORDERED);

    // Create resource
    createResource(eNS_URI);
  }

} //DevsPackageImpl
```

Implementación DevsAdapterFactory.java.

```
/**
*/
package devs.util;

import devs.*;

import org.eclipse.emf.common.notify.Adapter;
import org.eclipse.emf.common.notify.Notifier;

import org.eclipse.emf.common.notify.impl.AdapterFactoryImpl;

import org.eclipse.emf.ecore.EObject;

/**
* <!-- begin-user-doc -->
* The <b>Adapter Factory</b> for the model.
* It provides an adapter <code>createXXX</code> method for
* each class of the model.
* <!-- end-user-doc -->
* @see devs.DevsPackage
* @generated
*/
public class DevsAdapterFactory
 extends AdapterFactoryImpl {
  /**
  * The cached model package.
```

```
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
protected static DevsPackage modelPackage;

/**
* Creates an instance of the adapter factory.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @generated
*/
public DevsAdapterFactory() {
  if (modelPackage == null) {
    modelPackage = DevsPackage.eINSTANCE;
  }
}

/**
* Returns whether this factory is applicable for the
* type of the object.
* <!-- begin-user-doc -->
* This implementation returns <code>true</code> if the
* object is either the model's package or is an instance
* object of the model.
* <!-- end-user-doc -->
* @return whether this factory is applicable for the
* type of the object.
* @generated
*/
@Override
public boolean isFactoryForType(Object object) {
  if (object == modelPackage) {
    return true;
  }
  if (object instanceof EObject) {
    return ((EObject)object).eClass()
     .getEPackage() == modelPackage;
  }
  return false;
}

/**
* The switch that delegates to the
* <code>createXXX</code> methods.
* <!-- begin-user-doc -->
```

```
* <!-- end-user-doc -->
* @generated
*/
protected DevsSwitch<Adapter> modelSwitch =
 new DevsSwitch<Adapter>() {
  @Override
  public Adapter caseAtomicModel(AtomicModel object) {
    return createAtomicModelAdapter();
  }
  @Override
  public Adapter caseState(State object) {
    return createStateAdapter();
  }
  @Override
  public Adapter caseEvent(Event object) {
    return createEventAdapter();
  }
  @Override
  public Adapter caseInputEvent(InputEvent object) {
    return createInputEventAdapter();
  }
  @Override
  public Adapter caseOutputEvent(OutputEvent object) {
    return createOutputEventAdapter();
  }
  @Override
  public Adapter caseTransition(Transition object) {
    return createTransitionAdapter();
  }
  @Override
  public Adapter caseInternalTransition(
   InternalTransition object) {
    return createInternalTransitionAdapter();
  }
  @Override
  public Adapter caseExternalTransition(
   ExternalTransition object) {
    return createExternalTransitionAdapter();
  }
  @Override
  public Adapter caseOutputFunction(
   OutputFunction object) {
    return createOutputFunctionAdapter();
  }
  @Override
  public Adapter defaultCase(EObject object) {
```

```
    return createEObjectAdapter();
  }
};

/**
* Creates an adapter for the <code>target</code>.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @param target the object to adapt.
* @return the adapter for the <code>target</code>.
* @generated
*/
@Override
public Adapter createAdapter(Notifier target) {
  return modelSwitch.doSwitch((EObject)target);
}


/**
* Creates a new adapter for an object of class '{@link
* devs.AtomicModel <em>Atomic Model</em>}'.
* <!-- begin-user-doc -->
* This default implementation returns null so that we
* can easily ignore cases;
* it's useful to ignore a case when inheritance will
* catch all the cases anyway.
* <!-- end-user-doc -->
* @return the new adapter.
* @see devs.AtomicModel
* @generated
*/
public Adapter createAtomicModelAdapter() {
  return null;
}

/**
* Creates a new adapter for an object of class '{@link
* devs.State <em>State</em>}'.
* <!-- begin-user-doc -->
* This default implementation returns null so that we
* can easily ignore cases;
* it's useful to ignore a case when inheritance will
* catch all the cases anyway.
* <!-- end-user-doc -->
* @return the new adapter.
* @see devs.State
```

```
* @generated
*/
public Adapter createStateAdapter() {
  return null;
}

/**
* Creates a new adapter for an object of class '{@link
* devs.Event <em>Event</em>}'.
* <!-- begin-user-doc -->
* This default implementation returns null so that we
* can easily ignore cases;
* it's useful to ignore a case when inheritance will
* catch all the cases anyway.
* <!-- end-user-doc -->
* @return the new adapter.
* @see devs.Event
* @generated
*/
public Adapter createEventAdapter() {
  return null;
}

/**
* Creates a new adapter for an object of class '{@link
* devs.InputEvent <em>Input Event</em>}'.
* <!-- begin-user-doc -->
* This default implementation returns null so that we
* can easily ignore cases;
* it's useful to ignore a case when inheritance will
* catch all the cases anyway.
* <!-- end-user-doc -->
* @return the new adapter.
* @see devs.InputEvent
* @generated
*/
public Adapter createInputEventAdapter() {
  return null;
}

/**
* Creates a new adapter for an object of class '{@link
* devs.OutputEvent <em>Output Event</em>}'.
* <!-- begin-user-doc -->
* This default implementation returns null so that we
* can easily ignore cases;
```

```
* it's useful to ignore a case when inheritance will
* catch all the cases anyway.
* <!-- end-user-doc -->
* @return the new adapter.
* @see devs.OutputEvent
* @generated
*/
public Adapter createOutputEventAdapter() {
  return null;
}

/**
* Creates a new adapter for an object of class '{@link
* devs.Transition <em>Transition</em>}'.
* <!-- begin-user-doc -->
* This default implementation returns null so that we
* can easily ignore cases;
* it's useful to ignore a case when inheritance will
* catch all the cases anyway.
* <!-- end-user-doc -->
* @return the new adapter.
* @see devs.Transition
* @generated
*/
public Adapter createTransitionAdapter() {
 return null;
}

/**
* Creates a new adapter for an object of class '{@link
* devs.InternalTransition <em>Internal Transition</em>}'.
* <!-- begin-user-doc -->
* This default implementation returns null so that we
* can easily ignore cases;
* it's useful to ignore a case when inheritance will
* catch all the cases anyway.
* <!-- end-user-doc -->
* @return the new adapter.
* @see devs.InternalTransition
* @generated
*/
public Adapter createInternalTransitionAdapter() {
  return null;
}

/**
```

```
   * Creates a new adapter for an object of class '{@link
   * devs.ExternalTransition <em>External Transition</em>}'.
   * <!-- begin-user-doc -->
   * This default implementation returns null so that we
   * can easily ignore cases;
   * it's useful to ignore a case when inheritance will
   * catch all the cases anyway.
   * <!-- end-user-doc -->
   * @return the new adapter.
      * @see devs.ExternalTransition
   * @generated
   */
   public Adapter createExternalTransitionAdapter() {
     return null;
   }

   /**
   * Creates a new adapter for an object of class '{@link
   * devs.OutputFunction <em>Output Function</em>}'.
   * <!-- begin-user-doc -->
   * This default implementation returns null so that we
   * can easily ignore cases;
   * it's useful to ignore a case when inheritance will
   * catch all the cases anyway.
   * <!-- end-user-doc -->
   * @return the new adapter.
   * @see devs.OutputFunction
   * @generated
   */
   public Adapter createOutputFunctionAdapter() {
     return null;
   }

   /**
   * Creates a new adapter for the default case.
   * <!-- begin-user-doc -->
   * This default implementation returns null.
   * <!-- end-user-doc -->
   * @return the new adapter.
   * @generated
   */
   public Adapter createEObjectAdapter() {
     return null;
   }

 } //DevsAdapterFactory
```

Implementación DevsSwitch.java.

```
/**
*/
package devs.util;

import java.util.List;

import devs.*;

import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EObject;

/**
* <!-- begin-user-doc -->
* The <b>Switch</b> for the model's inheritance hierarchy.
* It supports the call {@link #doSwitch(EObject)
* doSwitch(object)}
* to invoke the <code>caseXXX</code> method for each class
* of the model,
* starting with the actual class of the object
* and proceeding up the inheritance hierarchy
* until a non-null result is returned,
* which is the result of the switch.
* <!-- end-user-doc -->
* @see devs.DevsPackage
* @generated
*/
public class DevsSwitch<T> {
  /**
  * The cached model package
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  protected static DevsPackage modelPackage;

  /**
  * Creates an instance of the switch.
  * <!-- begin-user-doc -->
  * <!-- end-user-doc -->
  * @generated
  */
  public DevsSwitch() {
```

```
  if (modelPackage == null) {
    modelPackage = DevsPackage.eINSTANCE;
  }
}

/**
* Calls <code>caseXXX</code> for each class of the model
* until one returns a non null result; it yields that
* result.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the first non-null result returned by a
* <code>caseXXX</code> call.
* @generated
*/
public T doSwitch(EObject theEObject) {
  return doSwitch(theEObject.eClass(), theEObject);
}

/**
* Calls <code>caseXXX</code> for each class of the model
* until one returns a non null result; it yields that
* result.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the first non-null result returned by a
* <code>caseXXX</code> call.
* @generated
*/
protected T doSwitch(EClass theEClass,
 EObject theEObject) {
  if (theEClass.eContainer() == modelPackage) {
    return doSwitch(theEClass.getClassifierID(),
     theEObject);
  }
  else {
    List<EClass> eSuperTypes =
     theEClass.getESuperTypes();
    return
     eSuperTypes.isEmpty() ?
      defaultCase(theEObject) :
       doSwitch(eSuperTypes.get(0), theEObject);
  }
}

/**
```

```
* Calls <code>caseXXX</code> for each class of the model
* until one returns a non null result; it yields that
* result.
* <!-- begin-user-doc -->
* <!-- end-user-doc -->
* @return the first non-null result returned by a
* <code>caseXXX</code> call.
* @generated
*/
protected T doSwitch(int classifierID,
 EObject theEObject) {
  switch (classifierID) {
    case DevsPackage.ATOMIC_MODEL: {
      AtomicModel atomicModel = (AtomicModel)theEObject;
      T result = caseAtomicModel(atomicModel);
      if (result == null) result =
       defaultCase(theEObject);
      return result;
    }
    case DevsPackage.STATE: {
      State state = (State)theEObject;
      T result = caseState(state);
      if (result == null) result =
       defaultCase(theEObject);
      return result;
    }
    case DevsPackage.EVENT: {
      Event event = (Event)theEObject;
      T result = caseEvent(event);
      if (result == null) result =
       defaultCase(theEObject);
      return result;
    }
    case DevsPackage.INPUT_EVENT: {
      InputEvent inputEvent = (InputEvent)theEObject;
      T result = caseInputEvent(inputEvent);
      if (result == null) result = caseEvent(inputEvent);
      if (result == null) result =
       defaultCase(theEObject);
      return result;
    }
    case DevsPackage.OUTPUT_EVENT: {
      OutputEvent outputEvent = (OutputEvent)theEObject;
      T result = caseOutputEvent(outputEvent);
      if (result == null) result =
       caseEvent(outputEvent);
```

```
        if (result == null) result =
         defaultCase(theEObject);
        return result;
      }
      case DevsPackage.TRANSITION: {
        Transition transition = (Transition)theEObject;
           T result = caseTransition(transition);
        if (result == null) result =
         defaultCase(theEObject);
        return result;
      }
        case DevsPackage.INTERNAL_TRANSITION: {
        InternalTransition internalTransition =
         (InternalTransition)theEObject;
        T result =
         caseInternalTransition(internalTransition);
        if (result == null) result =
         caseTransition(internalTransition);
        if (result == null) result =
         defaultCase(theEObject);
        return result;
      }
      case DevsPackage.EXTERNAL_TRANSITION: {
        ExternalTransition externalTransition =
         (ExternalTransition)theEObject;
        T result =
         caseExternalTransition(externalTransition);
        if (result == null) result =
         caseTransition(externalTransition);
        if (result == null) result =
         defaultCase(theEObject);
        return result;
      }
      case DevsPackage.OUTPUT_FUNCTION: {
        OutputFunction outputFunction =
         (OutputFunction)theEObject;
        T result = caseOutputFunction(outputFunction);
        if (result == null) result =
         defaultCase(theEObject);
        return result;
      }
      default: return defaultCase(theEObject);
    }
  }

  /**
```

```
* Returns the result of interpreting the object as an
* instance of '<em>Atomic Model</em>'.
* <!-- begin-user-doc -->
* This implementation returns null;
* returning a non-null result will terminate the switch.
* <!-- end-user-doc -->
* @param object the target of the switch.
* @return the result of interpreting the object as an
* instance of '<em>Atomic Model</em>'.
* @see #doSwitch(org.eclipse.emf.ecore.EObject)
* doSwitch(EObject)
* @generated
*/
public T caseAtomicModel(AtomicModel object) {
  return null;
}

/**
* Returns the result of interpreting the object as an
* instance of '<em>State</em>'.
* <!-- begin-user-doc -->
* This implementation returns null;
* returning a non-null result will terminate the switch.
* <!-- end-user-doc -->
* @param object the target of the switch.
* @return the result of interpreting the object as an
* instance of '<em>State</em>'.
* @see #doSwitch(org.eclipse.emf.ecore.EObject)
* doSwitch(EObject)
* @generated
*/
public T caseState(State object) {
  return null;
}

/**
* Returns the result of interpreting the object as an
* instance of '<em>Event</em>'.
* <!-- begin-user-doc -->
* This implementation returns null;
* returning a non-null result will terminate the switch.
* <!-- end-user-doc -->
* @param object the target of the switch.
* @return the result of interpreting the object as an
* instance of '<em>Event</em>'.
* @see #doSwitch(org.eclipse.emf.ecore.EObject)
```

```
* doSwitch(EObject)
* @generated
*/
public T caseEvent(Event object) {
  return null;
}

/**
* Returns the result of interpreting the object as an
* instance of '<em>Input Event</em>'.
* <!-- begin-user-doc -->
* This implementation returns null;
* returning a non-null result will terminate the switch.
* <!-- end-user-doc -->
* @param object the target of the switch.
* @return the result of interpreting the object as an
* instance of '<em>Input Event</em>'.
* @see #doSwitch(org.eclipse.emf.ecore.EObject)
* doSwitch(EObject)
* @generated
*/
public T caseInputEvent(InputEvent object) {
  return null;
}

/**
* Returns the result of interpreting the object as an
* instance of '<em>Output Event</em>'.
* <!-- begin-user-doc -->
* This implementation returns null;
* returning a non-null result will terminate the switch.
* <!-- end-user-doc -->
* @param object the target of the switch.
* @return the result of interpreting the object as an
* instance of '<em>Output Event</em>'.
* @see #doSwitch(org.eclipse.emf.ecore.EObject)
* doSwitch(EObject)
* @generated
*/
public T caseOutputEvent(OutputEvent object) {
  return null;
}

/**
* Returns the result of interpreting the object as an
* instance of '<em>Transition</em>'.
```

```
* <!-- begin-user-doc -->
* This implementation returns null;
* returning a non-null result will terminate the switch.
* <!-- end-user-doc -->
* @param object the target of the switch.
* @return the result of interpreting the object as an
* instance of '<em>Transition</em>'.
* @see #doSwitch(org.eclipse.emf.ecore.EObject)
* doSwitch(EObject)
* @generated
*/
public T caseTransition(Transition object) {
  return null;
}

/**
* Returns the result of interpreting the object as an
* instance of '<em>Internal Transition</em>'.
* <!-- begin-user-doc -->
* This implementation returns null;
* returning a non-null result will terminate the switch.
* <!-- end-user-doc -->
* @param object the target of the switch.
* @return the result of interpreting the object as an
* instance of '<em>Internal Transition</em>'.
* @see #doSwitch(org.eclipse.emf.ecore.EObject)
* doSwitch(EObject)
* @generated
*/
public T caseInternalTransition(
 InternalTransition object) {
  return null;
}

/**
* Returns the result of interpreting the object as an
* instance of '<em>External Transition</em>'.
* <!-- begin-user-doc -->
* This implementation returns null;
* returning a non-null result will terminate the switch.
* <!-- end-user-doc -->
* @param object the target of the switch.
* @return the result of interpreting the object as an
* instance of '<em>External Transition</em>'.
* @see #doSwitch(org.eclipse.emf.ecore.EObject)
* doSwitch(EObject)
```

```
  * @generated
  */
  public T caseExternalTransition(
   ExternalTransition object) {
    return null;
  }

  /**
  * Returns the result of interpreting the object as an
  * instance of '<em>Output Function</em>'.
  * <!-- begin-user-doc -->
  * This implementation returns null;
  * returning a non-null result will terminate the switch.
  * <!-- end-user-doc -->
  * @param object the target of the switch.
  * @return the result of interpreting the object as an
  * instance of '<em>Output Function</em>'.
  * @see #doSwitch(org.eclipse.emf.ecore.EObject)
  * doSwitch(EObject)
  * @generated
  */
  public T caseOutputFunction(OutputFunction object) {
    return null;
  }

  /**
  * Returns the result of interpreting the object as an
  * instance of '<em>EObject</em>'.
  * <!-- begin-user-doc -->
  * This implementation returns null;
  * returning a non-null result will terminate the switch,
  * but this is the last case anyway.
  * <!-- end-user-doc -->
  * @param object the target of the switch.
  * @return the result of interpreting the object as an
  * instance of '<em>EObject</em>'.
  * @see #doSwitch(org.eclipse.emf.ecore.EObject)
  * @generated
  */
  public T defaultCase(EObject object) {
    return null;
  }

} //DevsSwitch
```

2. Implementación de la clase UsingMediniQVTfromApplications.java para la inicialización del motor de Medini QVT-Relations.

```
/*
* Copyright (c) 2007 ikv++ technologies ag
* All rights reserved.
*/
package mediniTransformation;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.io.Reader;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;

import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EPackage;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceFactoryImpl;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceImpl;
import org.eclipse.uml2.uml.UMLPackage;

import uk.ac.kent.cs.kmf.util.ILog;
import uk.ac.kent.cs.kmf.util.OutputStreamLog;
import de.ikv.emf.qvt.EMFQvtProcessorImpl;
import de.ikv.medini.qvt.QVTProcessorConsts;
import devs.DevsPackage;

/**
* Class, which demonstrated how to use medini QVT
* inside your Java application. See the TODOs in
* the source code for assistance.
*
* @author Hajo Eichler <eichler@ikv.de>
*/
public class UsingMediniQVTfromApplications {
```

```
private ILog logger;

private EMFQvtProcessorImpl processorImpl;

protected ResourceSet resourceSet;

/**
* Initializes the QVT processor
*
* @param outputStream
*            an {@link OutputStream} for logging
*/
public UsingMediniQVTfromApplications(OutputStream outputStream) {
  this.logger = new OutputStreamLog(outputStream);
  this.processorImpl = new EMFQvtProcessorImpl(this.logger);
  this.processorImpl.setProperty(QVTProcessorConsts.PROP_DEBUG, "true");
}

/**
* Provide the information about the metamodels,
* which are involved in the transformation
*
* @param ePackages
*            the metamodel packages
*/
public void init(Collection<EPackage> ePackages) {
  Iterator<EPackage> iterator = ePackages.iterator();
  while (iterator.hasNext()) {
    this.processorImpl.addMetaModel(iterator.next());
  }
}

private void clean() {
  this.processorImpl.setModels(Collections.EMPTY_LIST);
}

/**
* Before transforming, the engine has to now the
* model to perform the transformation on, as
* well as a directory for the traces to store.
*
* @param modelResources
* models for the execution -
* take care of the right order!
* @param workingDirectory
```

```
* working directory of the QVT engine
*/
public void preExecution(Collection<?> modelResources, URI workingDirectory) {
  this.processorImpl.setWorkingLocation(workingDirectory);
  this.processorImpl.setModels(modelResources);
}

/**
* Transform a QVT script in a specific direction.
*
* @param qvtRuleSet
*            the QVT transformation
* @param transformation
*            name of the transformation
* @param direction
*  name of the target
*  - must conform to your QVT transformation
*  definition
* @throws Throwable
*/
public void transform(Reader qvtRuleSet,
  String transformation, String direction) throws Throwable {
    this.processorImpl.evaluateQVT(qvtRuleSet, transformation,
      true, direction, new Object[0], null, this.logger);
    this.clean();
  }

/**
* Helper for XMI loading. Does lazy loading.
*
* @param xmlFileName
*            file name to load
* @return the EMF resource
*/
public Resource getResource(String xmlFileName) {
  URI uri = URI.createFileURI(xmlFileName);
  Resource resource = null;
  try {
    resource = this.resourceSet.getResource(uri, true);
  } catch (Throwable fileNotFoundException) {
    resource = this.resourceSet.createResource(uri);
  }
  return resource;
}

/**
```

```
* Example usage of the class {@link UsingMediniQVTfromApplications}.
*
* @param args
*            are ignored here!
*/
public static void main(String[] args) {

  /*
  // initialize the engine
  UsingMediniQVTfromApplications simpleQVT =
    new UsingMediniQVTfromApplications(System.out);

  simpleQVT.launch();
  */
}

public void launch(String source, String target,String script,
  String traces,String ndirection, String ntransformation, String nameFile) {

    // initialize resource set of models
    this.resourceSet = new ResourceSetImpl();
    this.resourceSet.getResourceFactoryRegistry()
      .getExtensionToFactoryMap().put(
        Resource.Factory.Registry.DEFAULT_EXTENSION,
        new XMIResourceFactoryImpl());
        this.resourceSet = new ResourceSetImpl();
        this.resourceSet.getResourceFactoryRegistry()
          .getExtensionToFactoryMap().put(
            Resource.Factory.Registry.DEFAULT_EXTENSION,
            new ResourceFactoryImpl(){
              public Resource createResource(URI uri){
                return new XMIResourceImpl(uri){
                  protected boolean useUUIDs() {
                    return true;
                  }
                };
              }
            }
          );

    // TODO: collect all necessary packages from the metamodel(s)
    Collection<EPackage> metaPackages = new ArrayList<EPackage>();
    this.collectMetaModels(metaPackages);

    // make these packages known to the QVT engine
    this.init(metaPackages);
```

```
  // load the example model files - TODO: adjust the filename!
  Resource resource1 = this.getResource(source);
  Resource resource2 = this.getResource(target);


  // Collect the models, which should participate
  // in the transformation.
  // You can provide a list of models for each direction.
  // The models must be added in the same order as
  // defined in your transformation!
  Collection<Collection<Resource>> modelResources
    = new ArrayList<Collection<Resource>>();
  Collection<Resource> firstSetOfModels
    = new ArrayList<Resource>();
  Collection<Resource> secondSetOfModels
    = new ArrayList<Resource>();
  modelResources.add(firstSetOfModels);
  modelResources.add(secondSetOfModels);
  firstSetOfModels.add(resource1);
  secondSetOfModels.add(resource2);

  // tell the QVT engine a directory to work in
  // - e.g. to store the trace (meta)models
  URI directory = URI.createFileURI(traces);
  this.preExecution(modelResources, directory);

  // load the QVT relations
  FileReader qvtRuleSet = null;
  File filee2 = null;
  try {
    filee2= new File("SCC2SCS.qvt");
    FileWriter fileWriter = new FileWriter(filee2 , false);
    fileWriter.write(script);
    fileWriter.close();
    qvtRuleSet = new FileReader(filee2); // TODO: adjust the filename!
  } catch (FileNotFoundException fileNotFoundException) {
    fileNotFoundException.printStackTrace();
    return;
  } catch (IOException e){
    e.printStackTrace();
  } catch (Exception e){
    e.printStackTrace();
}
// tell the QVT engine, which transformation to
// execute - there might be more than one in
```

```
    // the QVT file!
    String transformation = ntransformation; // TODO: adjust
    // give the direction of the transformation
    // (according to the transformation definition)
    String direction = ndirection; // TODO: adjust

    // just do it ;-)
    try {
      this.transform(qvtRuleSet, transformation, direction);
    } catch (Throwable throwable) {
      throwable.printStackTrace();
    }

    // Note: the engine does not save the model
    // resources, which were participating in the
    // transformation.
    // You have to take care on this.
    try {
      resource2.save(Collections.EMPTY_MAP);
    } catch (IOException exception) {
      exception.printStackTrace();
    }
  }

  /**
  * Add all metamodel packages of models/qvt script
  *
  * @param metaPackages
  * @return
  */
  protected void collectMetaModels(Collection<EPackage> metaPackages) {
    metaPackages.add(UMLPackage.eINSTANCE);
    metaPackages.add(DevsPackage.eINSTANCE);
  }
}
```

Appendix C

Implementación Código Java para ejecutar transformaciones definidas con Acceleo (M2T)

Código completo para inicializar el motor de Acceleo y ejecutar la transformación M2T.

```
/*******************************************************************************
* Copyright (c) 2008, 2012 Obeo.
* All rights reserved. This program and the accompanying
* materials
* are made available under the terms of the Eclipse Public
* License v1.0
* which accompanies this distribution, and is available at
* http://www.eclipse.org/legal/epl-v10.html
*
* Contributors:
*     Obeo - initial API and implementation
*******************************************************************************/
package M2T.main;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.eclipse.acceleo.engine.event.IAcceleoTextGenerationListener;
import org.eclipse.acceleo.engine.generation
```

```
 .strategy.IAcceleoGenerationStrategy;
import org.eclipse.acceleo.engine.service.AbstractAcceleoGenerator;
import org.eclipse.emf.common.util.BasicMonitor;
import org.eclipse.emf.common.util.Monitor;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.resource.ResourceSet;

/**
* Entry point of the 'Generate' generation module.
*
* @generated
*/
public class Generate extends AbstractAcceleoGenerator {
  /**
  * The name of the module.
  *
  * @generated
  */
  public static final String MODULE_FILE_NAME =
   "/M2T/main/generate";

  /**
  * The name of the templates that are to be generated.
  *
  * @generated
  */
  public static final String[] TEMPLATE_NAMES = {
   "generate_h", "generate_pdm", "generate_cpp" };

  /**
  * The list of properties files from the launch parameters
   (Launch configuration).
  *
  * @generated
  */
  private List<String> propertiesFiles =
   new ArrayList<String>();

  /**
  * Allows the public constructor to be used. Note that a
   generator created
  * this way cannot be used to launch generations before one of
  * {@link #initialize(EObject, File, List)} or
  * {@link #initialize(URI, File, List)} is called.
  * <p>
```

```
* The main reason for this constructor is to allow clients
* of this
* generation to call it from another Java file, as it allows
* for the
* retrieval of {@link #getProperties()} and
* {@link #getGenerationListeners()}.
* </p>
*
* @generated
*/
public Generate() {
  // Empty implementation
}

/**
* This allows clients to instantiates a generator with all
* required information.
*
* @param modelURI
*            URI where the model on which this generator
* will be used is located.
* @param targetFolder
*            This will be used as the output folder for this
* generation : it will be the base path
*            against which all file block URLs will be
* resolved.
* @param arguments
*            If the template which will be called requires
* more than one argument taken from the model,
*            pass them here.
* @throws IOException
*             This can be thrown in three scenarios : the
* module cannot be found, it cannot be loaded, or
*             the model cannot be loaded.
* @generated
*/
public Generate(URI modelURI, File targetFolder,
 List<? extends Object> arguments) throws IOException {
  initialize(modelURI, targetFolder, arguments);
}

/**
* This allows clients to instantiates a generator with all
* required information.
*
* @param model
```

```
*            We'll iterate over the content of this element
* to find Objects matching the first parameter
*            of the template we need to call.
* @param targetFolder
*            This will be used as the output folder for this
* generation : it will be the base path
*            against which all file block URLs will be
* resolved.
* @param arguments
*            If the template which will be called requires
* more than one argument taken from the model,
*            pass them here.
* @throws IOException
*             This can be thrown in two scenarios : the
* module cannot be found, or it cannot be loaded.
* @generated
*/
public Generate(EObject model, File targetFolder,
 List<? extends Object> arguments) throws IOException {
  initialize(model, targetFolder, arguments);
}

/**
* This can be used to launch the generation from a
* standalone application.
*
* @param args
*            Arguments of the generation.
* @generated
*/
public static void main(String[] args) {
  try {
    if (args.length < 2) {
      System.out.println("Arguments not valid : {model,
       folder}.");
    } else {
      URI modelURI = URI.createFileURI(args[0]);
      File folder = new File(args[1]);

      List<String> arguments = new ArrayList<String>();

      /*
      * If you want to change the content of this method,
      * do NOT forget to change the "@generated"
      * tag in the Javadoc of this method
      * to "@generated NOT". Without this new tag, any
```

```
      * compilation
      * of the Acceleo module with the main template
      * that has caused the creation of this class will
      * revert your modifications.
      */

      /*
      * Add in this list all the arguments used by the
      * starting point of the generation
      * If your main template is called on an element of
      * your model and a String, you can
      * add in "arguments" this "String" attribute.
      */

      Generate generator = new Generate(modelURI, folder,
       arguments);

      /*
      * Add the properties from the launch arguments.
      * If you want to programmatically add new properties,
      * add them in "propertiesFiles"
      * You can add the absolute path of a properties files,
      * or even a project relative path.
      * If you want to add another "protocol" for your
      * properties files, please override
      * "getPropertiesLoaderService(AcceleoService)" in
      * order to return a new property loader.
      * The behavior of the properties loader service is
      * explained in the Acceleo documentation
      * (Help -> Help Contents).
      */

      for (int i = 2; i < args.length; i++) {
        generator.addPropertiesFile(args[i]);
      }

      generator.doGenerate(new BasicMonitor());
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}

/**
* Launches the generation described by this instance.
*
```

```
* @param monitor
*            This will be used to display progress
* information to the user.
* @throws IOException
*             This will be thrown if any of the output files
* cannot be saved to disk.
* @generated
*/
@Override
public void doGenerate(Monitor monitor) throws IOException {
  /*
  * TODO if you wish to change the generation as a whole,
  * override this. The default behavior should
  * be sufficient in most cases. If you want to change the
  * content of this method, do NOT forget to
  * change the "@generated" tag in the Javadoc of this
  * method to "@generated NOT". Without this new tag,
  * any compilation of the Acceleo module with the main
  * template that has caused the creation of this
  * class will revert your modifications. If you encounter a
  * problem with an unresolved proxy during the
  * generation, you can remove the comments in the following
  * instructions to check for problems. Please
  * note that those instructions may have a significant
  * impact on the performances.
  */

  //org.eclipse.emf.ecore.util.EcoreUtil.resolveAll(model);

  /*
  * If you want to check for potential errors in your models
  * before the launch of the generation, you
  * use the code below.
  */

  //if (model != null && model.eResource() != null) {
    //List<org.eclipse.emf.ecore.resource
     .Resource.Diagnostic> errors =
      model.eResource().getErrors();
    //for (org.eclipse.emf.ecore.resource
      .Resource.Diagnostic diagnostic : errors) {
       //System.err.println(diagnostic.toString());
    //}
 //}

  super.doGenerate(monitor);
```

```
}

/**
* If this generator needs to listen to text generation
* events, listeners can be returned from here.
*
* @return List of listeners that are to be notified when
* text is generated through this launch.
* @generated
*/
@Override
public List<IAcceleoTextGenerationListener>
 getGenerationListeners() {
  List<IAcceleoTextGenerationListener> listeners =
   super.getGenerationListeners();
  /*
  * TODO if you need to listen to generation event, add
  * listeners to the list here. If you want to change
  * the content of this method, do NOT forget to change the
  * "@generated" tag in the Javadoc of this method
  * to "@generated NOT". Without this new tag, any
  * compilation of the Acceleo module with the main template
  * that has caused the creation of this class will revert
  * your modifications.
  */
  return listeners;
}

/**
* If you need to change the way files are generated, this is
* your entry point.
* <p>
* The default is {@link
* org.eclipse.acceleo.engine.generation
*  .strategy.DefaultStrategy}; it generates
* files on the fly. If you only need to preview the results,
* return a new
* {@link
* org.eclipse.acceleo.engine
*  .generation.strategy.PreviewStrategy}. Both of these
* aren't aware of
* the running Eclipse and can be used standalone.
* </p>
* <p>
* If you need the file generation to be aware of the
* workspace (A typical example is when you wanna
```

```
* override files that are under clear case or any other VCS
* that could forbid the overriding), then
* return a new {@link
* org.eclipse.acceleo.engine
*  .generation.strategy.WorkspaceAwareStrategy}.
* <b>Note</b>, however, that this <b>cannot</b> be used
* standalone.
* </p>
* <p>
* All three of these default strategies support merging
* through JMerge.
* </p>
*
* @return The generation strategy that is to be used for
* generations launched through this launcher.
* @generated
*/
@Override
public IAcceleoGenerationStrategy getGenerationStrategy() {
  return super.getGenerationStrategy();
}

/**
* This will be called in order to find and load the module
* that will be launched through this launcher.
* We expect this name not to contain file extension, and the
* module to be located beside the launcher.
*
* @return The name of the module that is to be launched.
* @generated
*/
@Override
public String getModuleName() {
  return MODULE_FILE_NAME;
}

/**
* If the module(s) called by this launcher require
* properties files, return their qualified path from
* here.Take note that the first added properties files will
* take precedence over subsequent ones if they
* contain conflicting keys.
*
* @return The list of properties file we need to add to the
* generation context.
* @see java.util.ResourceBundle#getBundle(String)
```

```
* @generated
*/
@Override
public List<String> getProperties() {
  /*
  * If you want to change the content of this method, do NOT
  * forget to change the "@generated"
  * tag in the Javadoc of this method to "@generated NOT".
  * Without this new tag, any compilation
  * of the Acceleo module with the main template that has
  * caused the creation of this class will
  * revert your modifications.
  */

  /*
  * TODO if your generation module requires access to
  * properties files, add their qualified path to the list
  * here.
  *
  * Properties files can be located in an Eclipse plug-in or
  * in the file system (all Acceleo projects are Eclipse
  * plug-in). In order to use properties files located in an
  * Eclipse plugin, you need to add the path of the
  * properties
  * files to the "propertiesFiles" list:
  *
  * final String prefix = "platform:/plugin/";
  * final String pluginName =
  * "org.eclipse.acceleo.module.sample";
  * final String packagePath =
  * "/org/eclipse/acceleo/module/sample/properties/";
  * final String fileName = "default.properties";
  * propertiesFiles.add(prefix + pluginName + packagePath
  * + fileName);
  *
  * With this mechanism, you can load properties files from
  * your plugin or from another plugin.
  *
  * You may want to load properties files from the file
  * system, for that you need to add the absolute path of
  * the file:
  *
  * propertiesFiles.add("C:\Users\MyName\MyFile.properties");
  *
  * If you want to let your users add properties files
  * located in the same folder as the model:
```

```
  *
  * if (EMFPlugin.IS_ECLIPSE_RUNNING && model != null &&
  * model.eResource() != null) {
  * propertiesFiles.addAll(
  * AcceleoEngineUtils.getPropertiesFilesNearModel(
  * model.eResource()));
  * }
  *
  * To learn more about Properties Files, have a look at the
  * Acceleo documentation (Help -> Help Contents).
  */
    return propertiesFiles;
}

/**
* Adds a properties file in the list of properties files.
*
* @param propertiesFile
*            The properties file to add.
* @generated
* @since 3.1
*/
@Override
public void addPropertiesFile(String propertiesFile) {
  this.propertiesFiles.add(propertiesFile);
}

/**
* This will be used to get the list of templates that are to
* be launched by this launcher.
*
* @return The list of templates to call on the module {@link
* #getModuleName()}.
* @generated
*/
@Override
public String[] getTemplateNames() {
  return TEMPLATE_NAMES;
}

/**
* This can be used to update the resource set's package
* registry with all needed EPackages.
*
* @param resourceSet
*            The resource set which registry has to be *updated.
```

```
* @generated
*/
@Override
public void registerPackages(ResourceSet resourceSet) {
  super.registerPackages(resourceSet);

  /*
  * If you want to change the content of this method, do NOT
  * forget to change the "@generated"
  * tag in the Javadoc of this method to "@generated NOT".
  * Without this new tag, any compilation
  * of the Acceleo module with the main template that has
  * caused the creation of this class will
  * revert your modifications.
  */

  /*
  * If you need additional package registrations, you can
  * register them here. The following line
  * (in comment) is an example of the package
  * registration for
  * UML.
  *
  * You can use the method  "isInWorkspace(Class c)" to check
  * if the package that you are about to
  * register is in the workspace.
  *
  * To register a package properly, please follow the
  * following conventions:
  *
  * If the package is located in another plug-in, already
  * installed in Eclipse. The following content should
  * have been generated at the beginning of this method. Do
  * not register the package using this mechanism if
  * the metamodel is located in the workspace.
  *
  * if (!isInWorkspace(UMLPackage.class)) {
  *     // The normal package registration if your metamodel
  * is in a plugin.
  *     resourceSet.getPackageRegistry()
  *.put(UMLPackage.eNS_URI,
  * UMLPackage.eINSTANCE);
  * }
  *
  * If the package is located in another project in your
  * workspace, the plugin containing the package has not
```

```
  * been register by EMF and Acceleo should register it
  * automatically. If you want to use the generator in
  * stand alone, the regular registration (
  * seen a couple lines
  * before) is needed.
  *
  * To learn more about Package Registration, have a look at
  * the Acceleo documentation (Help -> Help Contents).
  */
}

/**
* This can be used to update the resource set's resource
* factory registry with all needed factories.
*
* @param resourceSet
*            The resource set which registry has to be
* updated.
* @generated
*/
@Override
public void registerResourceFactories(
 ResourceSet resourceSet) {
  super.registerResourceFactories(resourceSet);
  /*
  * If you want to change the content of this method, do NOT
  * forget to change the "@generated"
  * tag in the Javadoc of this method to "@generated NOT".
  * Without this new tag, any compilation
  * of the Acceleo module with the main template that has
  * caused the creation of this class will
  * revert your modifications.
  */

  /*
  * TODO If you need additional resource factories
  * registrations, you can register them here. the following
  * line
  * (in comment) is an example of the resource factory
  * registration for UML.
  *
  * If you want to use the generator in stand alone, the
  * resource factory registration will be required.
  *
  * To learn more about the registration of Resource
  * Factories, have a look at the Acceleo documentation
```

```
    * (Help -> Help Contents).
    */

    // resourceSet.getResourceFactoryRegistry()
    //  .getExtensionToFactoryMap().put(
    //  UMLResource.FILE_EXTENSION,
    //  UMLResource.Factory.INSTANCE);
  }

}
```