ESCUELA TÉCNICA SUPERIOR DE INGENIERIA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

TRABAJO FIN DE GRADO

Desarrollo de máquinas de estados jerárquicas en Java siguiendo un enfoque de desarrollo dirigido por modelos

AUTOR: FRANCISCO HERNANDEZ MARTINEZ

DIRECTOR: DIEGO ALONSO CÁCERES

SEPTIEMBRE 2012

Universidad Politécnica de Cartagena

Ficha de Propuesta Trabajo Fin de Grado

Departamento:	Tecnología de la Información y las Comunicaciones	
Curso académico:	2011/2012	
Fecha:	10 / 09 / 2012	
Tipo de Proyecto:	☐ **PFC** ☒ **TFG**	
Titulación	Grado en Ingeniería Telemática	**ERASMUS (SI/NO): NO**
Título del Proyecto: "Desarrollo de máquinas de estados jerárquicas en Java siguiendo un enfoque de desarrollo dirigido por modelos"		
Traducción del Título del Proyecto a lengua inglesa: "Hierarchical state-machine development in Java following a model-driven approach"		

Director/a/s del proyecto:	DIEGO ALONSO CÁCERES

Nombre del alumno:	FRANCISCO HERNÁNDEZ MARTÍNEZ
D.N.I:	23040244-V
Expediente Nº:	
Fecha de inicio del proyecto:	/ /

VºBº Director del Departamento
(Firma y Sello)

El Director del Proyecto
(firmar en todas las páginas)

Fdo.:

Fdo.:

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN 1

1.1 Justificación. 2

1.2 Objetivo. 2

2. MÁQUINAS DE ESTADO 5

2.1 Definición: Máquinas de Estado Finito y Redes de Petri. 5

2.2 Máquina de Moore. 9

2.3 Máquina de Mealy 10

2.4 Máquinas de estado finito según David Harel. 11

2.5 Estándar UML 2. 12

2.6 Ventajas de una Máquina de Estado Finito. 14

2.7 Desventajas de una Máquina de Estado Finito. 14

3. DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS. 17

3.1 Introducción. 17

3.2 Proceso de desarrollo de software dirigido por modelos. 18

3.3 Transformaciones de modelos 19

3.4 Visión de la OMG: Arquitectura Dirigida por Modelos. 21

3.5 Descripción del entorno de desarrollo 24

3.5.1 Eclipse 24

3.5.2 Eclipse Modeling Framework. 25

3.5.3 MOFScript. 26

4. MÁQUINAS DE ESTADO JERÁRQUICAS. 29

4.1 Meta-Modelo 29

4.2 Código MOFSCRIPT para realizar las transformaciones 31

4.2.1 Transiciones a Funciones utilizando orientación a objetos: Ts2FsOO. 31

4.2.2 Transiciones a Funcion utilizando orientación a objetos: Ts2FOO. 34

4.2.3 Patrón Estado 38

4.3 Modelos de Ejemplo. 40

5. CONCLUSIONES Y TRABAJOS FUTUROS. 43

6. ANEXOS 45

ANEXO A: Transformación M2T utilizando MOFScript. 45

ANEXO B: Código MOFSCRIPT Transf. M2T. 49

B. 1 Código MOFSCRIPT. Ts2FsOO. 49

B. 2 Código MOFSCRIPT. Ts2FOO. 63

B. 3 Código MOFSCRIPT. Patrón Estado. 77

B. 4 Ejemplo de simulación 92

7. BIBLIOGRAFÍA 95

1. INTRODUCCIÓN

Según el diccionario de la Real Academia de la Lengua Española, un modelo es un «esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento». Hasta hace relativamente poco tiempo la ingeniería del software no se había fijado en esta forma de desarrollo, que durante tanto tiempo y con tan buenos resultados lleva aplicándose a otras ramas de la ciencia, como la física o la química.

El uso de modelos para desarrollar programas se denomina genéricamente *Model Driven Engineering* (MDE) y representa «una aproximación esperanzadora para solucionar la incapacidad de los lenguajes de tercera generación de reducir la complejidad de las plataformas de implementación y de expresar conceptos del dominio de forma efectiva».

Si bien es cierto que el desarrollo basado en modelos no es algo nuevo, su aplicación no ha sido posible hasta que se han desarrollado las primeras herramientas que proporcionan el soporte necesario para su aplicación. En este punto ha desempeñado un papel destacado el *Object Management Group* (OMG), desarrollando un amplio conjunto de herramientas y definiendo la *Model Driven Architecture* (MDA).

MDE supone un salto en el nivel de abstracción utilizado hasta ahora para desarrollar software: el foco de atención del desarrollador cambia de los detalles concretos de implementación (*código*) a los conceptos importantes del dominio (*modelo*). Además, MDE promueve la utilización de herramientas de transformación de modelos para obtener (semi-) automáticamente otros modelos (con mayor grado de detalle por ejemplo) o representaciones textuales (por ejemplo, código de un lenguaje de programación). Este enfoque sería imposible de utilizar si la OMG no hubiera realizado la especificación de MOF v2.0 y sin el desarrollo de las correspondientes herramientas de Eclipse.

1.1 JUSTIFICACIÓN.

La gran mayoría de los estudios realizados llegan a la conclusión de que la forma más natural para describir el comportamiento de un sistema complejo es mediante la utilización de estados y eventos entre estos estados. Las máquinas de estado finito y sus correspondientes diagramas de estado constituyen un mecanismo natural para agrupar cada fragmento del sistema en un todo.

Haciendo referencia a la publicación de David Harel [**1**], el problema de las máquinas de estados finito es, entre otros, que no permiten una jerarquía entre estados, ni ortogonalidad. Para intentar solventar este problema, Harel define los $\mathrm{stateCharts}$:

Statecharts = Diagramas de estado + Profundidad +

Ortogonalidad + Comunicación broadcast;

Con la utilización de este nuevo concepto, es posible definir el comportamiento de un sistema complejo mediante un $\mathrm{stateCharts}$, pero estos $\mathrm{stateCharts}$, siguen estando en un formato gráfico, por lo que su funcionalidad se ve reducida a una imagen.

Gracias a los avances que se producen en el estándar UML 2, en especial la posibilidad de modelado, es posible generar modelos que representen las características de los $\mathrm{stateCharts}$, introducidos por Harel.

A partir de estos modelos, gracias a las transformaciones modelo a texto introducidas por MDE (*Desarrollo de software dirigido por modelos*), es posible la generación de texto.

Combinando esta posibilidad de transformar modelos en texto con los estudios de David Harel, es posible pasar de un sistema representado por un diagrama de estados o $\mathrm{stateCharts}$, en los que su funcionalidad se ve reducida a una imagen a un código que implemente el funcionamiento de este sistema. Siendo esta la finalidad del Trabajo Final de Grado.

1.2 OBJETIVO.

El objetivo de este Trabajo Fin de Grado es el de diseñar una aplicación siguiendo el enfoque de desarrollo dirigido por modelos para modelar máquinas de estado, y obtener posteriormente varias implementaciones en código Java. Para ello se hará uso de las herramientas disponibles para el entorno de desarrollo Eclipse.

En el capítulo 2 de este trabajo se realizará una introducción teórica que tratará el estado actual de la técnica, desarrollando una introducción a las máquinas de estado finito (*FSM*), las cuales se definen como modelos de comportamiento de un sistema o un objeto complejo, con un número limitado de modos o condiciones predefinidos, donde existen transiciones de modo. Según David Harel, se pueden definir como sistemas reactivos, los cuales están caracterizados por ser un sistema dirigido por eventos que continuamente reacciona a estímulos externos o internos. Como alternativa a las máquinas de estado finito, merece la pena destacar las redes de petri, sobre las que también se realizará una introducción. Una vez se ha realizado la introducción al mundo de las máquinas de estado, se analizarán los puntos de vista más importantes publicados sobre este tema, entre los que destacan las máquinas de Moore y de Mealy y más recientemente la aportación de David Harel y el estándar UML 2, siendo estas dos últimas visiones las utilizadas para la realización del

trabajo. Para finalizar esta introducción se destancan las ventajas e inconvenientes de utilizar máquinas de estados finitos.

En el apartado 3, se avanza un poco más en el desarrollo del estudio de máquinas de estado para hablar de la “Ingeniería Dirigida por Modelos”, a partir de ahora MDE, que surge para manejar el problema del crecimiento de la complejidad de los sistemas. Esta propuesta está centrada en modelos, en MDE cualquier concepto debe ser modelado. De esta manera, cualquier cambio o nueva propiedad del sistema debe ser mostrado en su modelo correspondiente. Con este paradigma, la parte de escritura de código es una parte más del proceso de construcción de sistemas (quizás la menos importante), la cual se sugiere que se realice automáticamente.

La presentación de MDE concluye con el análisis la “Arquitectura Dirigida por Modelos”, también llamada MDA, como una visión de la OMG para llevar a cabo la implementación de MDE.

Para finalizar con la parte teórica de este trabajo, se realiza una introducción a Eclipse, que es el entorno de desarrollo utilizado en el trabajo, prestando una espeacial atención a EMF y la herramienta MOFScript. EMF es una estructura de modelado y generación de código que facilita la construcción de herramientas y otras aplicaciones basadas en un modelo estructurado de datos, mientras que MOFScript es un proyecto que permite el desarrollo de herramientas y un marco de trabajo para dar soporte a transformaciones modelo a texto, como por ejemplo, soporte a la generación e implementación de código o documentación a partir de modelos. La herramienta MOFScript se encarga de dar un soporte al lenguaje MOFScript en términos de edición, análisis y ejecución obedeciendo los posibles estandares futuros.

Una vez realizada una introducción teórica, el capítulo 4 se dedica a detallar la parte práctica del trabajo, en la que se realizará un estudio sobre metamodelos de máquinas de estado jerárquica compleja. En este estudio, se realizarán diversas transformaciones modelo a texto, concretamente la transformación se realiza a código JAVA. Se realizan tres tipos de transformaciones sobre el metamodelo de máquinas de estado:

- Una primera transformación tiene como objetivo llevar a cabo una transformación que permita crear código JAVA en el que cada una de las posibles transiciones de la máquina de estados esté representada mediante el uso de una función.
- Como segunda opción, se realiza una transformación cuyo objetivo es la implementación de código JAVA, en el que se utilice una única función para representar las posibles transiciones de la máquina de estados, a diferencia de la transformación anterior, en la que se asociaba una función a cada una de las transiciones posibles.
- La tercera y última transformación se encarga de la implementación de código JAVA que siga las directrices de un patrón de diseño, en particular el patrón estado.

Para realizar estas transformaciones se utiliza la herramienta MOFScript y será necesario crear unos modelos, que deben cumplir las especificaciones de sus respectivos metamodelos, sobre los cuales se ejecutan las transformaciones. Cada transformación dispone de un simulador, a través del cuál se puede comprobar el correcto funcionamiento del modelo de máquina de estados utilizado.

Las máquinas de estado jerárquicas se componen de una gran cantidad de elementos, los cuales aumentan la funcionalidad de este tipo de máquinas, aumentando la complejidad de la transformación. Por estos motivos y para facilitar una mejor comprensión,se ha decidido

realizar varios ejemplos, cada uno con su respectivo metamodelo, en los que se irá aumentando, de forma gradual, la complejidad del modelo, es decir, se creará un primer modelo con unas características mínimas y se irán añadiendo nuevas características en los siguiente modelos.

Al final de este trabajo, se incluyen una serie de anexos indicados para ayudar al usuario a comprender mejor las tareas llevadas a cabo, incluyendo el código MOFSCRIPT utilizado en las transformaciones.

2. MÁQUINAS DE ESTADO

En este apartado se intenta que el usuario sea capaz de comprender el funcionamiento de las máquinas de estado finito, por lo que este capítulo se inicia con una definición de máquinas de estado en el que se incluirá una pequeña introducción a las redes de petri. Además de la definición, se verán algunas de las aportaciones más importantes realizadas al estudio de máquinas de estado finito, como pueden ser las máquinas de Moore y Mealy, el punto de vista de David Harel y el estándar UML 2. Concretamente, estas dos últimas ideas serán la base de este trabajo.

2.1 DEFINICIÓN: MÁQUINAS DE ESTADO FINITO Y REDES DE PETRI.

Las *Máquinas de Estados Finitos* (*FSM*), también conocidas como *Autómatas de Estados Finitos* (*FSA*), explicado de forma simple, son modelos de comportamiento de un sistema o un objeto complejo, con un número limitado de modos o condiciones predefinidos, donde existen transiciones de modo.

Las FSMs están compuestas por 4 elementos principales:

- **Estados** que definen el comportamiento y pueden producir acciones.
- **Transiciones** de estado que son movimientos de un estado a otro.
- **Reglas o condiciones** que deben cumplirse para permitir un cambio de estado.
- **Eventos** que permiten el lanzamiento de las reglas y permiten las transiciones. Estos eventos pueden ser externos o internos.

Una máquina de estados finitos debe tener un estado inicial que actúa de punto de comienzo, y un estado actual que recuerda el producto de la anterior transición de estado. Los eventos recibidos como entrada actúan como disparadores, que causan una evaluación de las reglas que gobiernan las transiciones del estado actual a otro estado.

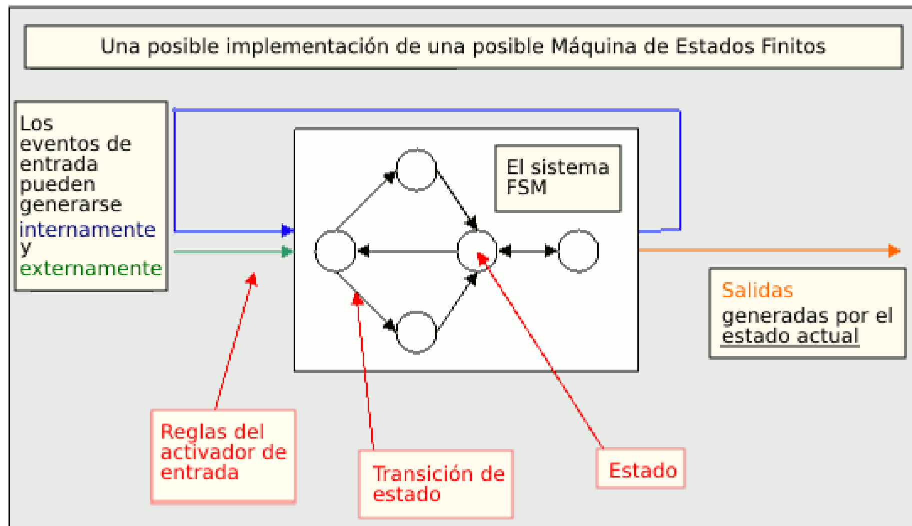


Figura 1. Posible implementación de una posible FSM

Las FSMs se usan típicamente como un tipo de sistema de control donde el conocimiento está representado en los estados, y las acciones están restringidas por las reglas. Es importante comprender la diferencia entre un estado y una acción. Se puede ver una **acción** como una actividad que consigue un objetivo como una evaluación o un movimiento, y un **estado** como una colección de acciones que se usan en un modo particular. Un estado es la circunstancia de algo, es una condición, y las acciones son atributos de ese estado, provee la habilidad de limitar el alcance de las acciones o la cantidad de conocimiento necesario para el estado actual.

Las máquinas de estados finitos son una técnica adoptada por la inteligencia artificial que se originó en el campo de las matemáticas, inicialmente utilizada para la representación de lenguajes. Como cualquier otro sistema basado en reglas, si todos los antecedente(s) de una regla son ciertos, entonces la regla se activa. Es posible que se activen múltiples reglas, y en el área de los sistemas de razonamiento, este hecho es conocido como grupo de conflicto. Sólo puede haber una transición desde el estado actual, por ello se requiere una estrategia consistente de resolución de conflictos para seleccionar una de las reglas activadas para disparar y así realizar la transición de estado [**2**].

Estas ideas brindan dos tipos principales de FSM. La simple FSM original es lo que se conoce como determinista, significando que dada una entrada y el estado actual, puede predecirse la transición de estado. Una extensión del concepto opuesto al anterior es una máquina de estados finitos no determinista. Dado el estado actual, la transición de estado no es predecible. Puede darse el caso que múltiples entradas se reciban en tipos diferentes, esto significaría que desde el estado actual no puede conocerse la transición a otro estado hasta que las entradas se reciban (dirigido por eventos).

Tradicionalmente, existen dos métodos principales para seleccionar donde generar las salidas de una máquina de estados finitos. Se les llaman Máquina de Moore y Máquina de

Mealy, llamadas así por el nombre de sus respectivos autores. Más adelante, se dedicará un apartado para hablar más detalladamente de este tipo de máquinas.

Antes de proseguir con lás máquinas de estado, daremos una breve introducción a las redes de Petri.

Como ya se ha comentado anteriormente, las redes de Petri representan una alternativa para modelar sistemas. Sus características hacen que para algunos problemas, las redes de Petri funcionen de una manera natural.

Las PN, que será como a partir de ahora se hará referencia a las redes de Petri (*Petri Net*), fueron inventadas por el alemán Karl Adam Petri en 1962. En su tesis doctoral "kommunikation mit automaten" (Comunicación con autómatas), establece los fundamentos para el desarrollo teórico de los conceptos básicos de las PN.

Las PN son consideradas una herramienta para el estudio de los sistemas. Con su ayuda se puede modelar el comportamiento y la estructura de un sistema, y llevar el modelo a condiciones límite, siendo esta situación muy difícil de lograr y/o muy costosa en un sistema real. Comparada con otros modelos de comportamiento dinámico gráficos, como los diagramas de las máquinas de estados finitos, las PN ofrecen una forma de expresar procesos que requieren sincronía, y quizás aún más importante, y es que las PN pueden ser analizadas de manera formal y obtener información del comportamiento dinámico del sistema modelado. Para modelar un sistema se usan representaciones matemáticas logrando una abstracción del sistema, esto es logrado con las PN, que además pueden ser estudiadas como autómatas e investigar sus propiedades matemáticas.

¿Qué tipo de sistemas se pueden modelar con las PN? Y ¿Cómo lograr la analogía entre el sistema real y el modelo usando una PN? son dos de las preguntas que se deben atender. Una idea fundamental en un sistema es que se compone de módulos que interactúan entre sí, los cuales pueden ser considerados por si mismos un sistema, y cuyo comportamiento podría ser estudiado por separado y de esta manera aislarlos, pero siempre teniendo en cuenta la interacción que guardan con los otros módulos.

Dos conceptos importates dentro de las redes de petri son los conceptos de acciones y estados. Las acciones conducen a un estado determinado del módulo en el tiempo, las acciones de un módulo en un sistema pueden ocurrir simultáneamente con las acciones de otros módulos, dado que ellos interacúan entre sí es necesario sincronizar los eventos. Esto puede resultar en que las condiciones de un módulo en el tiempo necesitan como entradas las salidas de otro, el cuál necesita más tiempo para generar las salidas, es entonces cuando entra en juego el paralelismo y la concurrencia. Las PN fueron diseñadas específicamente para modelar este tipo de sistemas.

Además de estos dos conceptos de acciones y estado, se han de tener en cuenta dos conceptos más: eventos y condiciones, los eventos son las acciones que se dan en el sistema y lo conducen a un estado, este estado se puede definir como un conjunto de condiciones. Para que cierto evento ocurra es necesario que ciertas condiciones se cumplan, estas son llamadas pre-condiciones del evento, la ocurrencia del evento puede llevar a otras condiciones y es entonces cuando se dan las post-condiciones.

Para modelar un sistema en una PN se deben conocer las condiciones y los eventos que se dan en él, de esta manera se puede hacer la analogía entre el sistema y el modelo. Al conocer las condiciones que se necesitan para dar cierto evento se pueden diseñar los módulos y relacionarlos con otras condiciones, y es por este motivo por lo que se necesita saber la estructura de una PN, para saber que corresponde a una condición y un evento en la red.

Las PN se componen de cuatro partes:

- Un conjunto de nodos.
- Un conjunto de transiciones.
- Una función de entrada y
- Una función de salida.

Las funciones de entrada y salida relacionan a los nodos y a las transiciones. La función de entrada es un mapeo de una transición t_j a una colección de nodos conocidos como los nodos de entrada de una transición. La estructura de una PN es definida por los nodos, las transiciones, la función de entrada y la función de salida.

Una caracterísitica a destacar es la asignación de tokens a la PN. Un token es un concepto primitivo de una PN, un número de ellos reside en los nodos y se mueve entre ellos; los tokens son la parte dinámica de la PN, su número puede variar entre nodos y son los que determinan la situación de la red en un momento determinado. La PN puede ser considerada también como un modelo de flujo de información, en donde el comportamiento dinámico de los tokens representa el flujo.

Una vez vistos los elementos que forman una red de petri, es importante saber realizar una representación gráfica de una PN, porque al observar el modelo del sistema en forma gráfica y observar como cambia de un estado a otro, se puede mantener la atención y dar una perspectiva más clara a quién esté analizando el problema:

- Un circulo O representa un nodo.
- Una barra | representa una transición.
- Los arcos o curvas conectan los nodos y las transiciones, si un arco va de un nodo a una transición, el nodo será una entrada y si el arco va de una transición a un nodo, el nodo será una salida de esa transición.
- Los tokens son representados por pequeños puntos •.

La ejecución en una PN es controlada por el número y distribución de los tokens que tiene. Los tokens presentes en los nodos controlan la ejecución de las transiciones de la red. Una PN se activa disparando transiciones. Una transición es disparada removiendo tokens de los nodos de entrada y creando tokens de salida. De aquí se obtiene la primera condición de disparo en una transición: *todos los nodos de entrada de la transición, deben tener al menos el mismo número de tokens que de arcos que van hacia la transición para que ésta sea disparada, cuando la transición cumpla esta condición se dice que es una transición ENABLED*.

Para realizar el modelado de sistemas complejos con PN, se divide el sistema en eventos y condiciones, para de esta manera encontrar la analogía con la PN. Para ello, se toma como referencia que las condiciones que se dan en un sistema son representadas por los nodos, ya que los tokens indican si esta condición se cumple o no, y los eventos son representados con las transiciones, que necesitan de condiciones para poder ser disparadas.

A partir de las ideas vistas, se puede concluir que:

- Debido a su facilidad de manejo en el problema de la sincronización de procesos, las redes de Petri son una alternativa de modelado de sistemas aplicados principalmente hacia el control y proceso,.

- Constan de cuatro partes: Nodos, transiciones, funciones de entrada y funciones de salida.
- Las entradas y/o salidas de una transición son conjuntos que pueden tener elementos repetidos o múltiples ocurrencias.
- Cuentan con una asignación de tokens que es la parte dinámica de las Redes de Petri.
- Las Redes de Petri se pueden representar gráficamente, un circulo O representa un nodo y una barra | representa una transición, y los tokens son representados por pequeños puntos •.
- Las Redes de Petri tienen reglas de disparo, siendo la principal, la que dice: "todos los nodos de entrada de la transición, deben tener al menos el mismo número de tokens que número de arcos van hacia la transición para que ésta sea disparada". Cuando la transición cumple dicha condición se dice que es ENABLED.
- Es posible modelar los sistemas dividiéndolos en eventos y condiciones. Las condiciones son representadas por los nodos, y los eventos por las transiciones.

Por último, merece la pena mencionar la existencia de extensiones a las Redes de Petri: por ejemplo las Redes de Petri Coloreadas (*PNC*), las Redes de Petri Temporales, Redes de Petri Estocásticas.

2.2 MÁQUINA DE MOORE.

Una “Máquina de Moore”, ver Figura 2, es un autómata de estados finitos donde las salidas están determinadas por el estado actual únicamente (y no depende directamente de la entrada). El Diagrama de estados para una máquina Moore incluirá una señal de salida para cada estado. Comparada con la Máquina de Mealy, la cual mapea transiciones en la maquina a salidas.

Una máquina de Moore puede ser definida como una 6-tupla {S, S0, Σ, Λ, T, G} consistente de

- un conjunto finito de estados (S)
- un estado inicio (también llamado estado inicial) S0 el cual es un elemento de (S)
- un conjunto finito llamado alfabeto entrada (Σ)
- un conjunto finito llamado el alfabeto salida (Λ)
- una función de transición (T : S × Σ → S) mapeando un estado y una entrada al siguiente estado
- una función salida (G: S → Λ) mapeando cada estado al alfabeto salida.

El número de estados en una máquina de Moore será mayor o igual al número de estados en la Máquina de Mealy correspondiente.

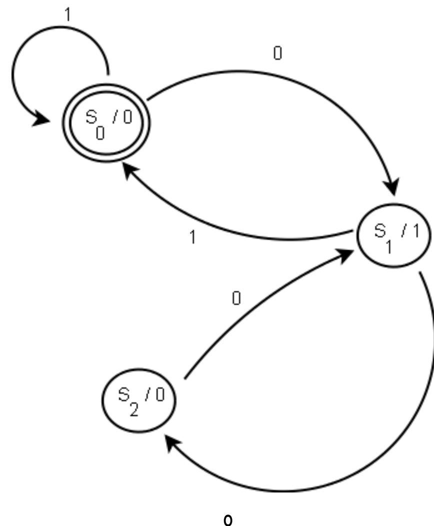


Figura 2. Máquina de Moore simple.

2.3 MÁQUINA DE MEALY

Una “Máquina de Mealy” es un tipo de máquina de estados finitos que genera una salida basándose en su estado actual y una entrada. Esto significa que el Diagrama de estados incluirá ambas señales de entrada y salida para cada línea de transición. En contraste, la salida de una máquina de Moore de estados finitos (el otro tipo) depende solo del estado actual de la máquina, dado que las transiciones no tienen entrada asociada. Sin embargo, para cada Máquina de Mealy hay una máquina de Moore equivalente cuyos estados son la unión de los estados de la máquina de Mealy y el Producto cartesiano de los estados de la máquina de Mealy y el alfabeto de entrada.

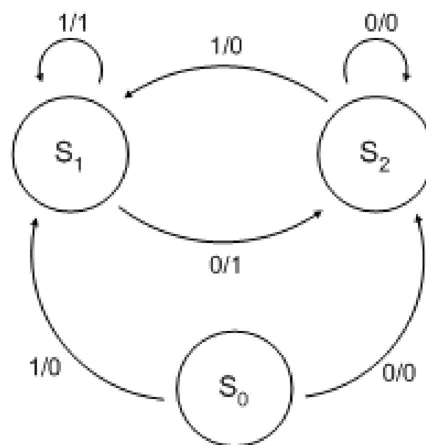


Figura 3. Máquina de Mealy simple

Una máquina de Mealy es una 6-tupla, (S, S0, Σ, Λ, T, G), consistiendo en

- un conjunto finito de estados (S)
- un estado inicial S0 el cual es un elemento de (S)
- un conjunto finito llamado el alfabeto entrada (Σ)

- un conjunto finito llamado el alfabeto salida (Λ)
- una función de transiciones (T : S × Σ → S)
- una función de salida (G : S × Σ → Λ)

2.4 MÁQUINAS DE ESTADO FINITO SEGÚN DAVID HAREL.

La literatura basada en software y en ingeniería de sistemas coincide en la existencia de un problema a la hora de realizar una descripción del comportamiento de un sistema reactivo complejo. Un sistema reactivo se caracteriza por estar dirigido por eventos, proporcionando una respuesta a estímulos tanto internos como externos.

Como se puede ver en [**1**], Harel expone que este problema esta en realizar una descripción de este comportamiento que sea clara y realista a la vez que formal y rigurosa. El comportamiento de un sistema reactivo es realmente un conjunto de secuencias de entrada y eventos de salida, condiciones y acciones, a veces con información adicional, como restricciones de tiempo.

En un sistema basado en transformaciones es suficiente con especificar una transformación que relacione la entrada con la salida, y aunque estos sistemas puedan llegar a ser bastante complejos, existen gran cantidad de métodos que permiten descomponer el comportamiento del sistema en varias partes coherentes y rigurosas al mismo tiempo. La mayoría de estas aproximaciones son soportadas por lenguajes de programación y herramientas de implementación, según Harel, este es un problema que todavía no ha sido resuelto para los sistemas reactivos, ya que aunque se han propuesto importantes y prometedoras aproximaciones, el sentimiento general es que hacen falta realizar bastantes mejoras y desarrollos.

La mayor parte de los estudios realizados, están de acuerdo en que el uso de estados y eventos son la mejor manera de describir el comportamiento de un sistema complejo. Las máquinas de estado finito así como sus correspondientes diagramas de transición-estado, constituyen un buen mecanismo para recoger en un mismo diagrama el comportamiento del sistema. Los diagramas de estado son simples gráficos que utilizan nodos para representar estados y flechas para representar transiciones. En la Figura 4 se puede ver un diagrama de estados simple.

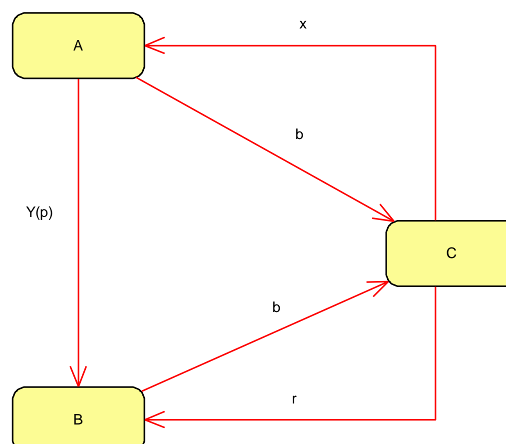


Figura 4. Diagrama de estados simple

Sin embargo, se puede ver claramente que un sistema complejo no podrá ser descrito correctamente a partir de un diagrama de estados de este tipo. Para que un diagrama de estados pueda llegar a ser provechoso, debe ser modular, jerárquico y estar bien estructurado. Cumpliendo estas características, se resolvería el problema que causaría un crecimiento exponencial de los elementos del sistema. Dicho esto, un buen diagrama de estados debería cumplir las siguientes clausulas:

- Capacidad de agrupar estados en un superestado.
- Capaz de introducir ortogonalidad e independencia.
- Alusión a una posible necesidad de tener varias transiciones generales, en lugar de una única transición.
- Perfeccionamiento de los estados.

David Harel presenta los statecharts como una posible solución a estos problemas. Los statecharts son un formalismo visual para describir estados y transiciones de una forma modular, proporcionando agrupación, ortogonalidad y perfeccionamiento, proporcionando un una posibilidad de 'zoom' para poder navegar con facilidad a través de los niveles de abstracción.

Técnicamente hablando, la base de esta aproximación consiste en la extensión de los diagramas de estados convencionales utilizando descomposición de estados a través de sentencias lógicas AND/OR, junto con transiciones entre niveles y un mecanismo de comunicación broadcast que permite una comunicación concurrente entre los componentes. Las dos ideas esenciales de esta extensión son la provisión de una profundidad de descripción y la noción de ortogonalidad. Esto se puede resumir en:

Statecharts = Diagramas de estado + Profundidad +

Ortogonalidad + Comunicación broadcast;

El desarrollo del trabajo se realizará basándose en la utilización de statecharts.

2.5 ESTÁNDAR UML 2.

Lenguaje Unificado de Modelado (*UML, por sus siglas en inglés, Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; aún cuando todavía no es un estándar oficial, está respaldado por el OMG (*Object Management Group*) [**3**]. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran

variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado de Rational) -pero no especifica en sí mismo qué metodología o proceso usar.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas. En UML 2.0 hay 13 tipos diferentes de diagramas. Para comprenderlos de manera concreta, se acompaña la **Figura 5**, en la que se pueden ver estos diagramas categorizados de forma jerárquica, así como una pequeña explicación de la función de cada diagrama:

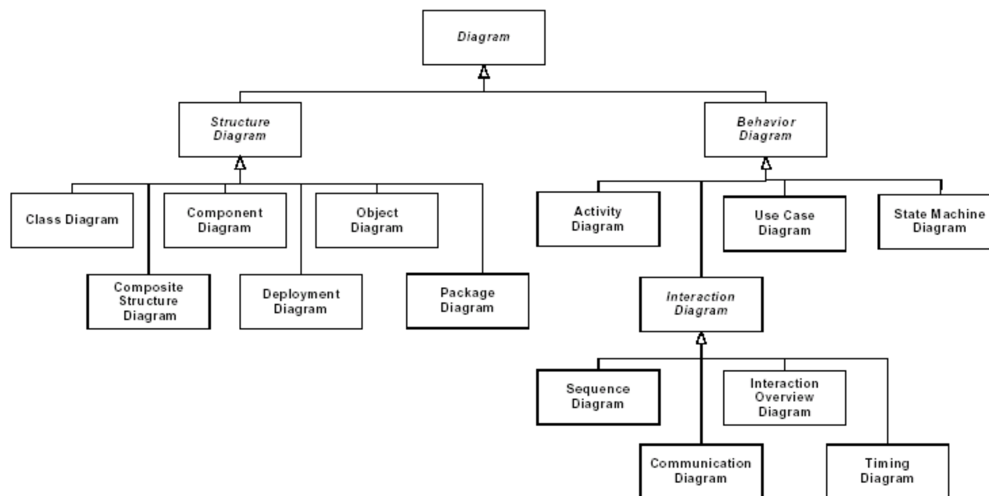


Figura 5. Estructura UML 2.

Diagramas de estructura enfatizan en los elementos que deben existir en el sistema modelado:

- Diagrama de clases
- Diagrama de componentes
- Diagrama de objetos
- Diagrama de estructura compuesta (UML 2.0)
- Diagrama de despliegue
- Diagrama de paquetes

Diagramas de comportamiento enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de actividades
- Diagrama de casos de uso
- Diagrama de estados

Diagramas de Interacción, ES un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de secuencia

- Diagrama de colaboración
- Diagrama de tiempos (UML 2.0)
- Diagrama de vista de interacción (UML 2.0)

2.6 VENTAJAS DE UNA MÁQUINA DE ESTADO FINITO.

- Su simplicidad hace fácil para los desarrolladores sin experiencia realizar la implementación con poco o nada de conocimiento extra (fácil entrada).
- Predictibilidad (en FSM deterministas), dado un grupo de entradas y un estado actual conocido, puede predecirse la transición de estados, facilitando la tarea de verificación.
- Dada su simplicidad, las FSM son rápidas de diseñar, rápidas de implementar y rápidas de ejecutar.
- FSM en una técnica antigua de representación de conocimiento y modelado de sistemas, ha sido usada desde hace tiempo y como tal, ha sido verificada como una técnica de inteligencia artificial, con muchos ejemplos de los que aprender.
- Las FSM son relativamente flexibles. Existen varias maneras de implementar un sistema basado en FSMs en términos de su topología, y es fácil incorporar muchas otras técnicas.
- La transferencia desde una representación abstracta del conocimiento a una implementación es fácil.
- Bajo uso del procesador; apropiado para dominios donde el tiempo de ejecución está compartido entre varios módulos o subsistemas. Solo el código del estado actual ha de ser ejecutado, además de un poco de lógica para determinar el estado actual.
- Es fácil determinar si se puede llegar o no a un estado. En las representaciones abstractas, resulta obvio si se puede o no llegar a un estado desde otro, y que requerimientos existen para hacerlo

2.7 DESVENTAJAS DE UNA MÁQUINA DE ESTADO FINITO.

- La naturaleza predecible de las FSM deterministas puede no resultar conveniente en algunos dominios, como los juegos por ordenador (la solución pasa por implementar una FSM no determinista).
- Si se implementa un sistema grande usando FSMs puede ser difícil de administrar y mantener sin un buen diseño. Las transiciones entre estados pueden causar cierto grado de "factor espagueti" al intentar seguir una línea de ejecución.
- No es apropiado para todos los dominios de problema, solo debe ser usado cuando el comportamiento de un sistema puede ser descompuesto en estados separados con condiciones bien definidas para las transiciones. Esto significa que todos los estados, transiciones y condiciones deben ser conocidos y estar bien definidos.

- Las condiciones para las transiciones entre estados son rígidas, es decir, están fijadas.

Como la mayoría de técnicas, las heurísticas para saber dónde y cómo implementar máquinas de estados finitos son subjetivas y dependen de cada problema específico. Está claro que las FSMs están bien adaptadas a dominios de problemas que se expresan fácilmente usando diagramas de flujo y poseen un grupo de estados y reglas que gobiernan las transiciones entre estados bien definidos.

3. Desarrollo de software dirigido por modelos.

Antes de comenzar a explicar el paradigma "Model-Driven Engineering", a partir de ahora MDE, es necesario dar una definición de los conceptos de metamodelo y modelo. Una vez definidos estos conceptos se comienza el estudio de MDE, haciendo hincapié en Model-Driven Architecture, a partir de ahora MDA, creado por el *Object Management Group* (*OMG*) como ejemplo de manera de llevar a cabo MDE.

3.1 INTRODUCCIÓN.

En este apartado, simplemente se dará una definición de los conceptos metamodelo y modelo. Un metamodelo se define como un conjunto finito de conceptos que se quieren modelar, más un conjunto de relaciones entre ellos. En la Figura 6 se puede observar un ejemplo de metamodelo.

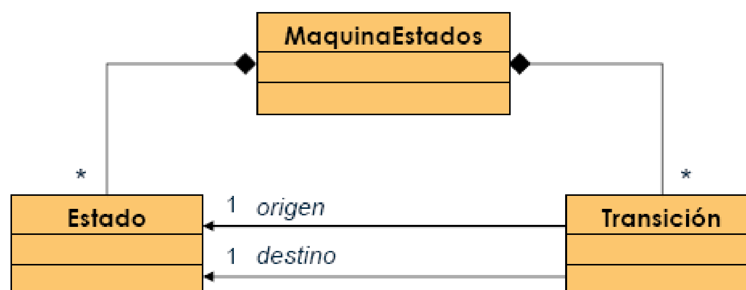


Figura 6. Ejemplo de MetaModelo

A partir de este metamodelo surge el lenguaje de modelado, que es el conjunto infinito de todos los modelos válidos que se puede crear a partir de un metamodelo. En la Figura 7, se pueden obsevar varios ejemplos de modelos:

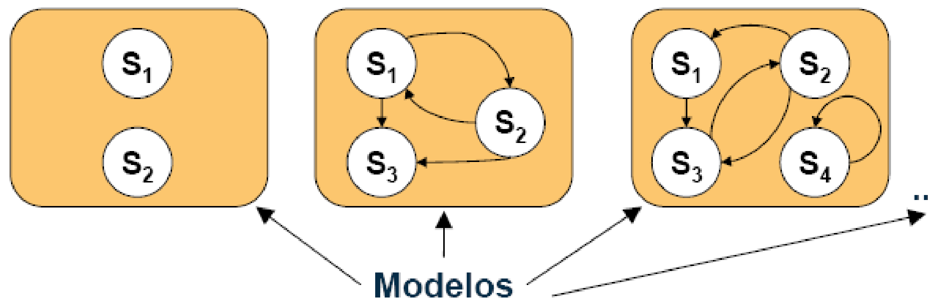


Figura 7. Modelos

Se puede definir un modelo como una representación abstracta del objeto, sistema o cualquier cosa que se desee crear.

3.2 PROCESO DE DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS.

Durante las últimas dos décadas, los avances en lenguajes y plataformas han aumentado el nivel de abstracción disponible en la tarea de desarrollo de software. Además, debido a la madurez de los lenguajes de tercera generación, los desarrolladores de software están mejor equipados para afrontar y resolver los distintos problemas que se les pueden plantear.

A pesar de todos estos avances, aún quedan problemas importantes que resolver. En el centro de éstos se encuentra el crecimiento de la complejidad de las plataformas, las cuales contienen miles de clases y métodos con dependencias muy complicadas que deben ser conocidas por el desarrollador. El problema crece cuando estas mismas plataformas crecen rápidamente y además aparecen otras nuevas, con el consiguiente esfuerzo de migración de unas a otras.

En este último caso, cuando la evolución tecnológica de las plataformas o de los sistemas se produce, es importante conservar el mismo modelo conceptual del negocio, es decir, que la lógica del dominio del problema debería ser la misma, sea cual sea la plataforma o lenguaje que implementa dicha lógica.

Para manejar el problema del crecimiento de la complejidad de los sistemas la orientación a objetos no parece ser suficiente. Los lenguajes orientados a objetos han ido perdiendo la simplicidad con la que fueron ideados, la encapsulación no es un recurso tan útil como en principio parecía y, sobre todo, la reutilización de los objetos como componentes no ha tenido demasiado éxito en la industria del software [**4**].

Parece que las propuestas centradas en código no dan respuesta a las demandas de los sistemas actuales. Esta es la razón por la que ha aparecido una nueva propuesta centrada en modelos.

A esta propuesta se le llama *Ingeniería Dirigida por Modelos* (*MDE*). Este paradigma combina los siguientes conceptos [**5**]:

- **Lenguajes de dominio específico:** Formalizan la estructura de la aplicación, el comportamiento y los requisitos dentro de un dominio particular. Estos lenguajes

(*DSL*) son descritos usando metamodelos, los cuales definen relaciones entre elementos dentro de un dominio.

- **Motores de transformación y generadores**: Analizan ciertos aspectos de los modelos, después crean varios tipos de artefactos, tal como código fuente, entradas de simulación, descripciones de uso XML, o representaciones alternativas de dicho modelo.

Las herramientas MDE usan los conceptos anteriores y hacen más fácil para los ingenieros del software el soporte a la evolución del software, tanto en su lógica como en su tecnología.

Mediante los DSLs se consiguen notaciones de modelado distintas para cada tipo de sistema, las cuales están definidas formalmente por su metamodelo. De esta manera, el ingeniero del software tiene herramientas específicas para cada tipo de sistema, lo cual le permite modelarlos de una manera más detallada y de acuerdo al dominio al que pertenecen.

Mediante los motores de transformación se facilita la evolución de modelos, transformando de unos modelos a otros, según la reglas de transformación entre metamodelos.

En el paradigma MDE cualquier concepto debe ser modelado. De esta manera, cualquier cambio o nueva propiedad del sistema debe ser mostrado en su modelo correspondiente. Con este paradigma, la parte de escritura de código es una parte más del proceso de construcción de sistemas (quizás la menos importante), la cual se sugiere que se realice automáticamente.

3.3 TRANSFORMACIONES DE MODELOS

La transformación de modelos es el proceso de convertir un modelo en otro modelo del mismo sistema.

Desde el punto de vista de MDE, los modelos evolucionan mediante transformaciones definidas entre los correspondientes meta-modelos. Estas transformaciones pueden ser:

- Modelo a Modelo (*M2M*): Generación de modelos a partir de otros modelos.
    - Horizontales
    - Verticales

- Modelo a Texto (*M2T*): El proyecto M2T, en el que se basa este trabajo final de grado, se centra en la generación de texto a partir de modelos. Sus objetivos son:
    - Proporcionar la puesta en marcha de estándares de la industria y estándar de Eclipse para motores M2T.
    - Proporcionar ejemplos de herramientas de desarrollo y una infraestructura común. Entre los proyectos más actuales podemos destacar:
        - JET: Proporciona entorno de generación de código y soluciones utilizadas por EMF.
        - Xpand: Lenguaje de plantillas de tipo estático que ofrece extensiones funcionales, transformación de modelos, validación de modelos y mucho más. También incluye un editor con características de color de sintaxis y señalización de errores entre otras.

- M2T Core: Entorno que permite a los clientes invocar soluciones model-to-text, independientes del lenguanje model-to-text.
- M2T Shared: Infraestructura de componentes compartidos entre diferentes lenguajes model-to-text.

La última version de M2T data de finales de Junio de 2010.

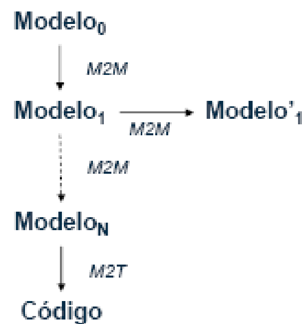


Figura 8. Tipos de transformaciones

En el paradigma MDA, la transformación de modelos puede ser horizontal o vertical. Una transformación horizontal consiste en pasar un modelo de un nivel de abstracción Mx a otro modelo del mismo nivel Mx, pero ambos basados en un modelo del nivel inmediato superior Mx-1 diferentes. Una transformación vertical ocurre cuando los modelos pertenecen a dos niveles inmediatos diferentes, Mx y Mx-1. Por ejemplo, se puede usar la transformación vertical de modelos para pasar de un modelo PIM a un modelo PSM, o de un modelo CIM a un modelo PIM; o una transformación horizontal para pasar de un PIM1 a otro PIM2 basado en un metamodelo diferente que el primero.

Las distintas etapas del ciclo de vida del software pueden ser representadas en función de los distintos tipos de modelos que MDA propone y de la transformación de dichos modelos con la transición de una etapa a otra. Para el caso en que se aplique el paradigma MDA a un sistema heredado (legado o antiguo), se puede aplicar la reingeniería para soportar su evolución, para lo que es necesario la transformación inversa de modelos (de PSM a PIM), lo cual también es sugerido por MDA.

El lenguaje estándar que OMG propone para la definición de transformaciones de modelos es el lenguaje *Query, View, Transformation* (*QVT*) [**6**], que a su vez se basa en el lenguaje de restricciones OCL (*Object Constraint Language*). Con QVT se pueden definir transformaciones genéricas entre metamodelos, así cualquier instancia del metamodelo fuente puede ser transformado en una instancia del metamodelo destino.

Otro tipo de transformación que merece la pena destacar es *Atlas Transformation Language* (ATL), desarrollado por el grupo ATLAS como respuesta al RFP-M2M del OMG.

Las características principales de ATL son:

- Lenguaje mixto: imperativo / declarativo.
- En ATL se pueden definir tres tipos de ficheros:
  - Module: transformación modelo a modelo.

- Library: funciones auxiliares reutilizables.
- Query: devuelve los elementos del modelo que cumplen determinadas propiedasde o restrcciones.

- Permite transformaciones MIMO.
- Dos tipos de transformación: normal y refinamiento.
- Tiene un depurador de transformaciones.
- Distingue mayúsculas y minúsculas.
- Comprobación de tipos en tiempo de ejecución. Difícil de depurar.

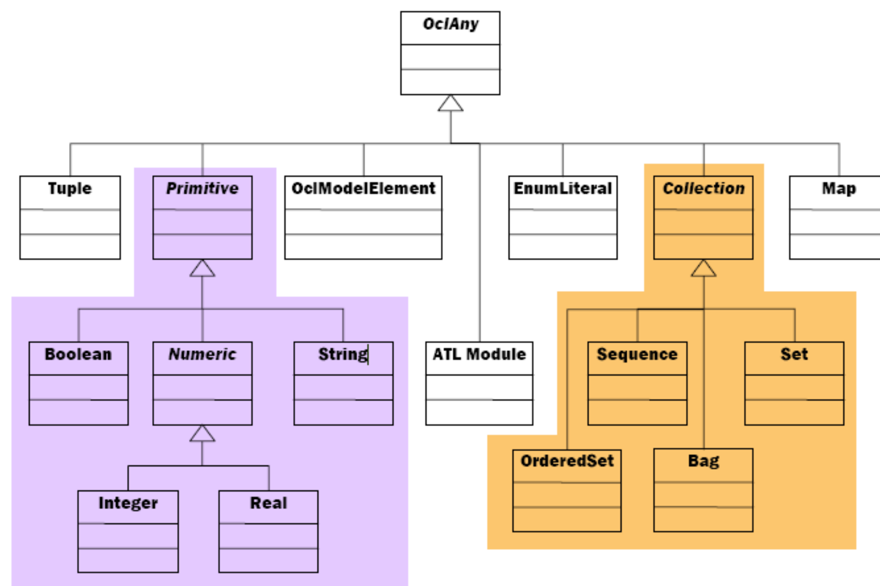


Figura 9. Tipos de datos en ATL.

3.4 VISIÓN DE LA OMG: ARQUITECTURA DIRIGIDA POR MODELOS.

El consorcio OMG (*Object Management Group*) ha desarrollado la propuesta "Arquitectura Dirigida por Modelos" o MDA como ejemplo de implementación de MDE.

Tal y como se puede extraer de [**7**], MDA (ver Figura 10) nace con la idea establecida de separar la especificación de la lógica operacional de un sistema, de los detalles que definen cómo el sistema usa las capacidades de la plataforma tecnológica donde es implementado.

Teniendo en cuenta lo anterior, los objetivos de MDA son la portabilidad, la interoperabilidad y la reusabilidad a través de la separación arquitectural.

El concepto de *independencia de plataforma* aparece frecuentemente en MDA. Es la cualidad que tienen los modelos de ser independientes de las características de cualquier tipo de plataforma tecnológica.

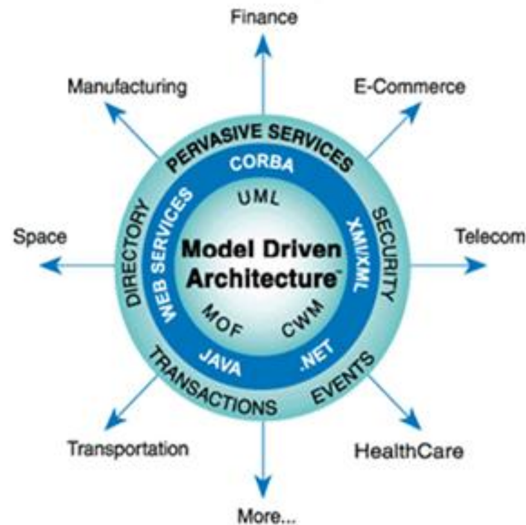


Figura 10. Esquema gráfico que representa las áreas y tecnologías que abarca MDA.

Mediante la aplicación de este paradigma se cubre completamente el ciclo de vida de un sistema software, desde la captura de requisitos hasta el mantenimiento del mismo, pasando por la generación del código fuente. Para ello define tres tipos de modelos que se explican a continuación, ver Figura 11:

- **Modelo Independiente de la Computación (CIM):** Un CIM no muestra detalles de la estructura del sistema. A veces es llamado modelo de dominio o modelo de negocio. En el CIM se modelan los requisitos que deberá satisfacer el sistema, describiendo la situación en la cual el sistema será usado. Es muy útil tanto para ayudar a comprender el problema como para ejercer de fuente de vocabulario compartido para el uso en otros modelos. Según MDA, la especificación de requisitos de un sistema CIM debería ser transformable en un PIM y posteriormente en un PSM y viceversa. El CIM juega un papel importante como puente entre los que son unos expertos en el dominio del problema y sus requisitos y aquellos que son expertos en el diseño y construcción de artefactos software.
- **Modelo Independiente de la Plataforma (PIM):** Un modelo PIM muestra el grado de independencia de plataforma necesario para poder ser usado en diferentes plataformas tecnológicas de un tipo similar. Este modelo debe tener tal nivel de abstracción que no cambie, sea cual sea la plataforma elegida para su implementación. Con este modelo se representa la lógica del sistema y sus interacciones con el mundo exterior, sin entrar en detalle de que tipo de tecnología implementará cada parte y cómo se adapta a una plataforma específica.
- **Modelo Específico de Plataforma (PSM)**: El modelo PSM es una vista del sistema para una plataforma específica. Éste combina la especificación del sistema hecha en el PIM, con los detalles que especifican la manera en que dicho sistema usa una plataforma particular.

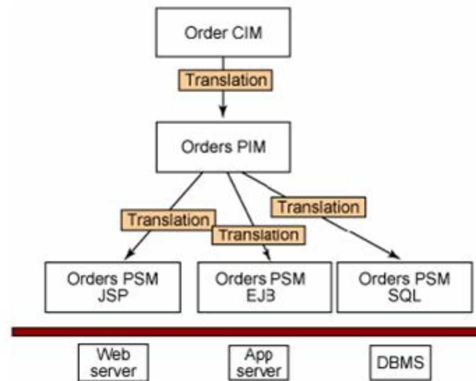


Figura 11. Modelos CIM, PIM y PSM y su relación de transformación.

Aunque UML, es el lenguaje de modelado central de MDA, no todos los modelos tienen por qué estar especificados en dicho lenguaje. Para ello entra en juego el concepto de metamodelo. Como se ha comentado anteriormente, un metamodelo es un modelo para definir modelos. UML es un metamodelo que especifica cómo crear modelos UML. Es decir, un modelo UML es una instancia del metamodelo UML.

Como se ha dicho, MDA no implica el uso de UML, pero en su lugar la tecnología crucial es MOF y la definición de metamodelos que sean instancias del metametamodelo MOF[**8**].

Cada uno de estos metamodelos define un lenguaje de modelado de dominio específico, que presenta una solución al modelado de distintos tipos de sistemas software. Por ejemplo, existe el metamodelo UML para modelar la arquitectura de sistemas discretos orientados a objetos, o el metamodelo SPEM para modelar procesos software, etc.

Todos estos metamodelos son a su vez instancias del meta-metamodelo MOF. Para entender bien este concepto en la Figura 12 se muestra la pirámide de niveles de modelado de la arquitectura conceptual de MOF.

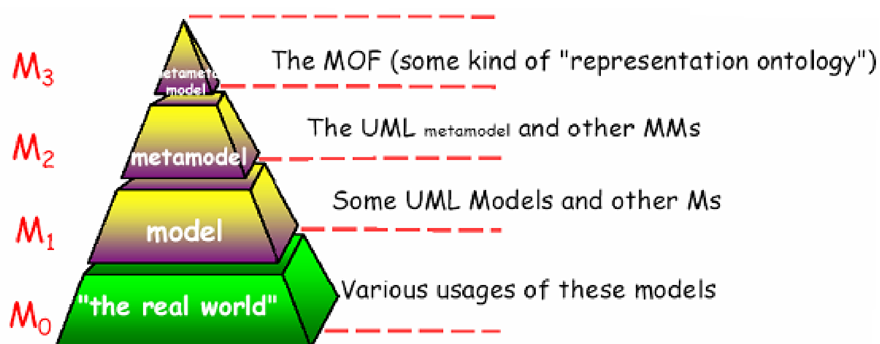


Figura 12. Pirámide de metamodelos y modelos MOF.

3.5 DESCRIPCIÓN DEL ENTORNO DE DESARROLLO

El entorno de desarrollo elegido para la realización del trabajo es Eclipse. Dentro de Eclipse, las herramientas utilizadas para realizar las transformaciones modelo a texto son EMF, que permite trabajar con modelos y la herramienta MOFSCRIPT, que da soporte para la generación de texto a partir de modelos. Para poder utilizar ambas herramientas será necesarias instalarlas como plugins de Eclipse. Estos plugins pueden obtenerse de la página web oficial de Eclipse (http:www.eclipse.org). Este capítulo esta destinado a ofrecer una introducción a Eclipse, así como a las herramientas de Eclipse utilizadas.

Resaltar que aunque existen otros entornos de desarrollo como pueden ser MetaEdit o DSLTools, se ha decidido optar por Eclipse, siendo los principales motivos que es un software de libre de distribución y sobre todo porque los demás entornos no presentan el concepto de "metamodelo".

3.5.1 Eclipse

Eclipse es una plataforma abierta y de libre distribución. Puede ser descargado de forma gratuita, al igual que todos sus plugins desde su página web, a través de la cual se puede acceder a gran cantidad de información sobre Eclipse.

http://www.eclipse.org

Además del uso de su página oficial, si se desea realizar un estudio más a fondo sobre Eclipse, se recomienda hacer uso de la referencia [**9**]. En su desarrollo participan importantes empresas como Borlan, IBM, Intel, Motorola,etc. Actualmente, la comunidad Eclipse se organiza en torno a múltiples proyectos que evolucionan en paralelo de manera independiente o cooperativa, estando los proyectos relacionados con la *Ingeniería dirigida por Modelos* (*MDE*) entre los más activos: EMF, GMF,M2M, M2T…

Entre las principales características de Eclipse, se pueden destacar las siguientes:

- Necesita run-time de Java (jre).
- No necesita instalación (se descomprime en cualquier carpeta).
- Fácil de extender con distintos plugins:
  - Se descargan y descomprimen directamente en el directorio \eclipse.
  - O mediante la opción de menú "Help→Software UPdate".

El entorno de trabajo de Eclipse está formado por:

- WorkSpace: directorio donde se almacenan todos los proyectos relacionados. Mantienen sus propias propiedades.
- View: ventanas de utilidad, como gestor de proyectos, propiedades, consola…
- Perspective: agrupación de vistas (views) que facilitan alguna tarea concreta, como por ejemplo desarrollo Java.

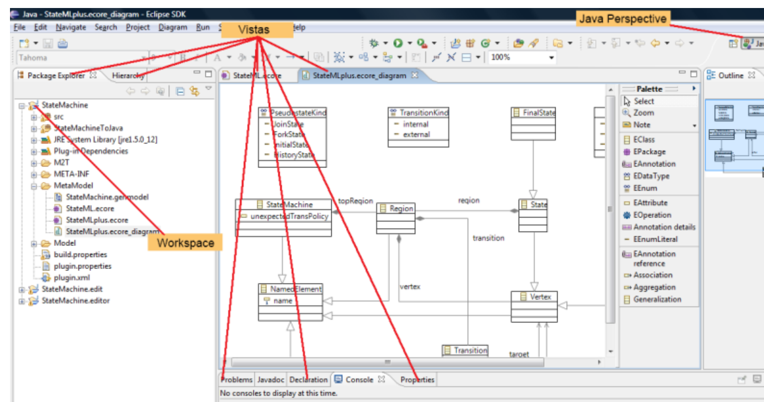


Figura 13. Plataforma Eclipse

En la siguiente figura se puede observar la arquitectura de Eclipse.

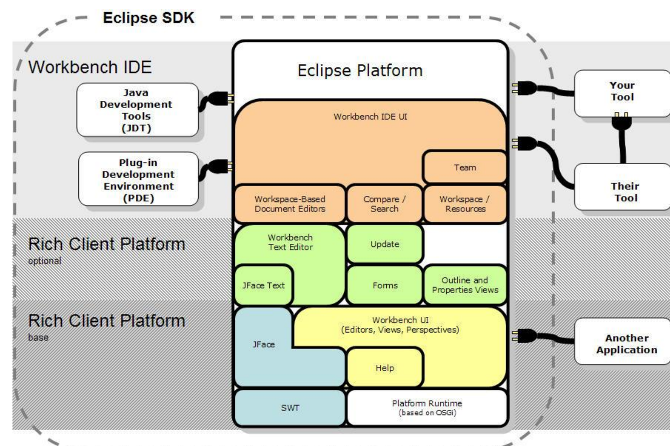


Figura 14. Arquitectura de Eclipse

3.5.2 *Eclipse Modeling Framework.*

La información utilizada para dar una introducción a EMF ha sido recogida de la página web oficial de eclipse (http://www.eclipse.org/emf).

El proyecto EMF (*Eclipse Modeling Framework*) es una estructura de modelado y generación de código que facilita la construcción de herramientas y otras aplicaciones basadas en un modelo estructurado de datos. De una especificación del modelo descrito en XMI, EMF proporciona las herramientas y el soporte al tiempo de ejecución para producir un conjunto de clases de código Java para el modelo, junto con un conjunto de las clases del adaptador que permiten la visión y la edición basada en comandos del modelo, y un editor básico. Los modelos pueden ser desarrollados usando notación Java, UML, documentos XML o herramientas de modelado como Rational Rose, después se importan a EMF. Lo más importante de todo es que EMF proporciona la posibilidad de interoperabilidad con otras herramientas y aplicaciones basadas en EMF.

EMF incluye *XML Schema Infoset Model* (*XSD*), un componente del proyecto *Model Deveplopment Tools* (*MDT*) y una implementación de *Service Data Objects* (*SDO*) basada en EMF. Otros subproyectos, se empaquetan por separado. XSD proporciona un

modelo y una API para la manipulación de componentes de un esquema XML, con acceso a la representación DOM subyacente del esquema.

EMF está formado por tres partes fundamentales:

- **EMF:** El núcleo de EMF incluye un metamodelo (Ecore) para describir modelos y dar soporte en tiempo de ejecución para los modelos, incluyendo notificación de cambios, dar soporte persistente con la serialización por defecto XMI y un API reflectivo muy eficiente para la manipulación de objetos EMF genéricos.
- **EMF.Edit:** La estructura EMF.Edit incluye clases reutilizables genéricas para la construcción de editores para modelos EMF. Proporciona:
    - Contenido y etiquetado de las clases proporcionadas y otras clases que permite a los modelos EMF ser expuestos usando visualizadores y hojas características del escritorio estándar (JFace).
    - Una estructura de commandos, incluyendo un conjunto de implementaciones de clases de commandos para la construcción de editors que soportan completamente las acciones de deshacer y rehacer automático.
- **EMF.Codegen:** La facilidad de la generación de código EMF es capaz de generar todo lo necesario para construir un completo editor para un modelo EMF. Incluye una GUI para la cual, las opciones de generación pueden ser especificadas y los generadores invocados. La facilidad de generación influye en el componente JDT (*Java Development Tooling*) de Eclipse.

Soporta tres niveles de generación de código:

- **Modelo:** Proporciona interfaces Java e implementación de clases para todas las clases en el modelo, añadiendo un paquete (meta datos) de clases implementadas.
- **Adaptadores:** Generan implementación de clases (llamadas ItemProviders) que adaptan las clases del modelo para la edición y exposición.
- **Editor:** Proporciona un editor correctamente estructurado que se ajusta a los estilos y servicios recomendados por los editores de modelo de Eclipse EMF, como un punto de partida a partir del cual, comenzar a realizar los cambios necesarios para cumplir con requisitos personales.

Todos los generadores soportan la regeneración de código conservando las modificaciones del usuario. Los generadores pueden ser invocados a través del GUI o desde la línea de comandos.

3.5.3 MOFScript.

Los objetivos del subproyecto MOFScript son el desarrollo de herramientas y estructuras que den soporte a las transformaciones modelo a texto, es decir, dar un soporte a la implementación de la generación de código o a la documentación de los modelos. Debería proveer una estructura de metamodelo neutra, que permita la utilización de algún tipo de metamodelo y sus instancias, para la generación de código.

El lenguaje MOFScript es una de las opciones del proceso OMG RFP en la transformación modelo a texto en MOF. Los objetivos del subproyecto MOFScript son dar un soporte al lenguaje MOFScript en términos de edición, análisis y ejecución obedeciendo los posibles estandares futuros.

MOFScript cubrirá los aspectos necesarios en el contexto de generación de texto en la ingeniería del software, incluyendo:

- Mecanismos de control: Capacidad de especificar control de mecanismo básico, de repetición (bucles) y de selección (mediante condiciones).
- Manipulación de string: Capacidad para manipular valores de string.
- Elementos del modelo referenciados por expresiones de salida.
- Creación de recursos de salida (ficheros): Capacidad de especificar el archivo para la generación de texto.
- Rastreabilidad entre modelos y texto generado: Capacidad de generar y utilizar rastreabilidad entre modelos fuente y texto generado, es decir, proporcionar regeneración.
- Facilidad de uso: Una interfaz fácil de usar.

La ingeniería inversa no es tratada todavía, pero podría ser incluida en un futuro.

DISPONIBILIDAD CÓDIGO MOFSCRIPT / CONTRIBUCIÓN INICIAL.

Se ha desarrollado un código que da soporte al análisis, chequeo, edición y ejecución de código MOFScript. Actualmente, este código puede encontrarse en plug-in separados, los cuales se ejecutan dentro de Eclipse. El programa de análisis y de tiempo de ejecución puede ser usado independientemente de Eclipse.

El siguiente código está listo para integrarse en el subproyecto MOFScript de Eclipse:

- **Módulo modelo MOFScript:** El modelo MOFScript es un modelo EMF/ecore, el cual es objetivo del programa análisis y el tema del tiempo de ejecución.
- **Módulo de análisis (Parser):** El programa de análisis lee el texto MOFScript y produce una instancia del modelo, la cual ha sido comprobada semánticamente.
- **Módulo tiempo de ejecución (Runtime):** El tiempo de ejecución proporciona un escenario de ejecución para un modelo MOFScript. Maneja todos los aspectos de la ejecución.
- **Módulo editor:** El editor proporciona un escenario del editor de Eclipse para el lenguaje MOFScript, incluyendo la ayuda, preferencias, documentación, etc.

DESCRIPCIÓN DE LA ARQUITECTURA MOFSCRIPT

La arquitectura de la herramienta MOFScript está formada por dos componentes lógicos principales: herramientas y servicios (ver Figura 15). El componente herramientas se refiere a las herramientas de usuario final, las cuales proporcionan capacidades de edición e interacción con los servicios. Los servicios proporcionan capacidades para el análisis, chequeo, y ejecución del lenguaje de transformación. El lenguaje está representado por un modelo (el modelo MOFScript), un modelo EMF popularizado por el programa de análisis. Este modelo es la base para el chequeo semántico y la ejecución. MOFScript es implementada como un plug-in de Eclipse usando el plug-in EMF para el manejo de modelos y metamodelos.

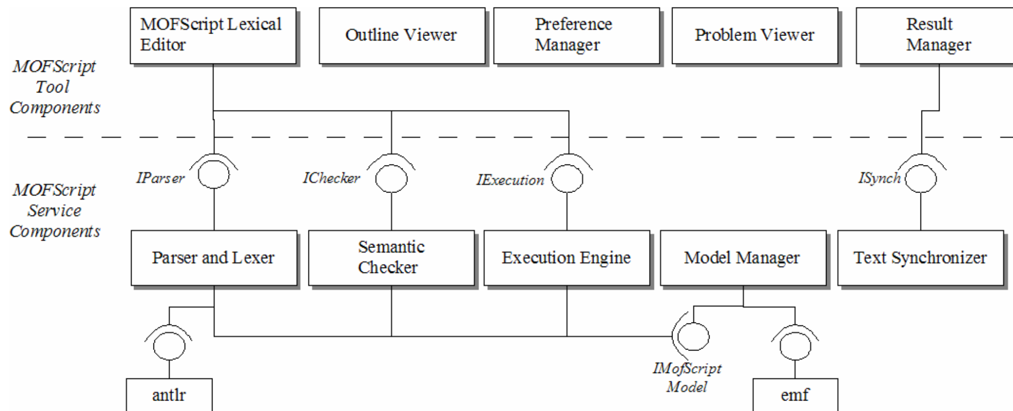


Figura 15. Herramientas y componentes de la arquitectura MOFSCRIPT

El componente servicios está formado por las siguientes partes:

- El *Model Manager* es un componente basado EMF, el cual se encarga de gestionar los modelos MOFScript.
- El *parser* y el *lexer* son responsables de analizar definiciones textuales de las transformaciones MOFScript y publicar un modelo MOFScript usando el Model Manager. El programa de análisis (parser) está basado en antlr.
- El *Semantic Checker* proporciona funcionalidad para comprobar la corrección de una transformación con respecto a la validez de las reglas utilizadas, referencias a elementos del metamodelo, etc.
- El *Execution Engine* maneja la ejecución de una transformación. Interpreta un modelo y produce una salida de texto.
- El *Text Synchroniser* maneja la traceabilidad entre el texto generado y el modelo original, con el objetivo de sincronizar el texto en respuesta a cambios en el modelo y viceversa [**10**].

En el anexo A, se detallan los pasos a seguir para realizar la transformación modelo a texto, utilizando MOFScript.

4. MÁQUINAS DE ESTADO JERÁRQUICAS.

Este capítulo está dedicado al estudio de máquinas de estado jerárquicas. El capítulo se inicia con una introducción al metamodelo utilizado, para seguidamente explicar el tipo de transformaciones que se realizarán, así como el formato en el que vendrán los resultados, en forma de clases e interfaces, una vez se hayan ejecutado cada una de las transformaciones sobre modelos que sigan este metamodelo. Estas explicaciones van acompañadas de sus respectivos diagramas de clases y de secuencia. Como referencia a la hora de tomar decisiones sobre el modo de implementar estas clases e interfaces se ha utilizado [**11**].

4.1 META-MODELO

El meta-modelo utilizado corresponde a una máquina de estados jerárquica con las siguientes características:

- La máquina de estados estará formada por regiones, dentro de las cuales estarán los distintos estados, pseudoestados y transiciones. Este metamodelo, también permitirá que un estado contenga regiones. A su vez, estas regiones contenidas en estados, estarán formadas por más transiciones y estados, que a su vez podrán tener regiones y así sucesivamente.
- Tipos de pseudoestados:
    - Histórico: Se encarga de recordar el último estado que se ha visitado en una región. De esta forma, la próxima vez que se entre a una región con pseudoEstado histórico, el estado actual de dicha región será el último visitado.

- Fork: A partir de este pseudoEstado, se lanzan varias transiciones. Se debe tener en cuenta que estas transiciones deben tener como destino distintas regiones, estando estas regiones dentro de un mismo estado, ya que no se permite que la máquina de estados esté en varios estados a la vez dentro de una misma región.
- Join: Este pseudoEstado sólo lanzará su transición asociada cuando se cumpla la condición de que la máquina de estados se encuentre en unos estados específicos. Tener en cuenta que estos estados estarán en distintas regiones dentro de un mismo estado.

- A la hora de crear el modelo, el usuario podrá solicitar el tipo de política que desea para tratar las transiciones que no pueden ejecutarse, dando un valor al atributo "**UnexpectedTransPolicy**".Los valores de este atributo pueden ser:
  - **null o ignore**: Cuando se intente realizar una transición no permitida, por ejemplo por estar en el estado fuente de dicha transición, simplemente lo mostrará por pantalla, de modo meramente informativo.
  - **throwException:** Al elegir esta opción, al intentar realizar una transición no permitida, además de mostrarlo por pantalla, se lanzará una excepción del tipo transitionErrorException.

Estas características obligan a tener un código complejo que pueda tener en cuenta las siguientes posibles situaciones:

- Se debe tener en cuenta que al **entrar a una nueva región** se debe comprobar si dicha región tiene un pseudoEstado de tipo inicial o de tipo histórico.
- Igualmente, al **salir de una región**, se debe comprobar si dicha región contiene un pseudoEstado de tipo histórico, actualizando su valor en caso de que lo contenga.
- Antes de **salir de un estado**, se debe salir de todas las regiones contenidas en este estado.
- Al ejecutar una **transición desde un pseudoEstado de tipo Join**, se debe comprobar que la máquina de estados se encuentra en los estados requeridos para poder lanzar esta transición.
- Al ejecutar una **transición con objetivo un pseudoEstado de tipo Fork**, esta transición se dividirá en varias transiciones, cada una de ellas con un destino diferente. Estos destinos deben ser distintas regiones de un mismo estado.

En la Figura 16, se puede ver el metamodelo explicado en modo gráfico.Una vez vistas las características de este nuevo metamodelo, ha llegado la hora de realizar las transformaciones que permitan convertir modelos que cumplan estas normas, teniendo en cuenta las nuevas características del metamodelo.

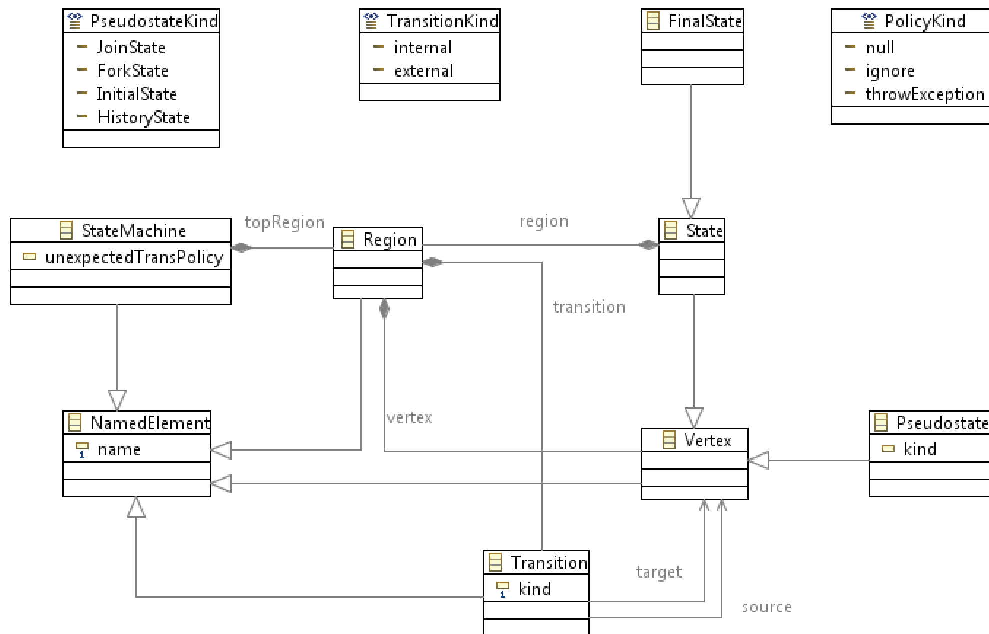


Figura 16. Meta-Modelo FSM Jerárquica.

4.2 CÓDIGO MOFSCRIPT PARA REALIZAR LAS TRANSFORMACIONES

Para llevar a cabo el estudio del metamodelo anterior, se realizarán tres transformaciones a código JAVA de modelos creados a partir de dicho metamodelo. Para realizar estas transformaciones se hará uso de código MOFSCRIPT, en el anexo A se puede ver una explicación de cómo crear y ejecutar un fichero MOFSCRIPT para realizar la transformación.

4.2.1 Transiciones a Funciones utilizando orientación a objetos: Ts2FsOO.

En esta primera transformación, se realizará una transformación a código JAVA, que convierta cada posible transición en una función.

Para realizar la transformación se ha realizado un estudio previo sobre si era más ventajoso la utilización de enumerados o una programación orientada a objetos. En resumen, se pueden observar una serie de ventajas que ofrece utilizar orientación a objetos. Una de las ventajas es que se permite almacenar el estado actual en una referencia a un objeto de tipo "State". A la hora de cambiar de estado, se modifica el valor de esta referencia por el valor de un objeto que represente al estado al que se pasa, pero realizando las mismas llamadas a función. Si no se utilizase orientación a objetos, al almacenar el estado actual en un tipo enumerado de datos, cada vez que se modifique el estado actual, se debe modificar también la llamada a función, lo que podría suponer varios problemas a la hora de realizar las transiciones que suponen salir o entrar a regiones, y es aquí donde se encuentran las mayores ventajas, ya que si se utilizase por ejemplo un tipo enumerado para guardar el estado actual de cada región, al intentar salir de varias regiónes no se puede invocar su función de salida sobre un tipo de datos enumerado, por lo que habría que realizar un código complejo para encontrar estos estados de los que se debe salir en cada región. Si se utiliza

orientación a objetos, si se puede invocar estas funciones de salida sobre los estados actuales de cada región, ya que están almacenados en objetos de tipo State.

Debido a las ventajas comentadas, se ha optado por una transformación orientada a objetos, dando como resultado el siguiente diagrama de clases:

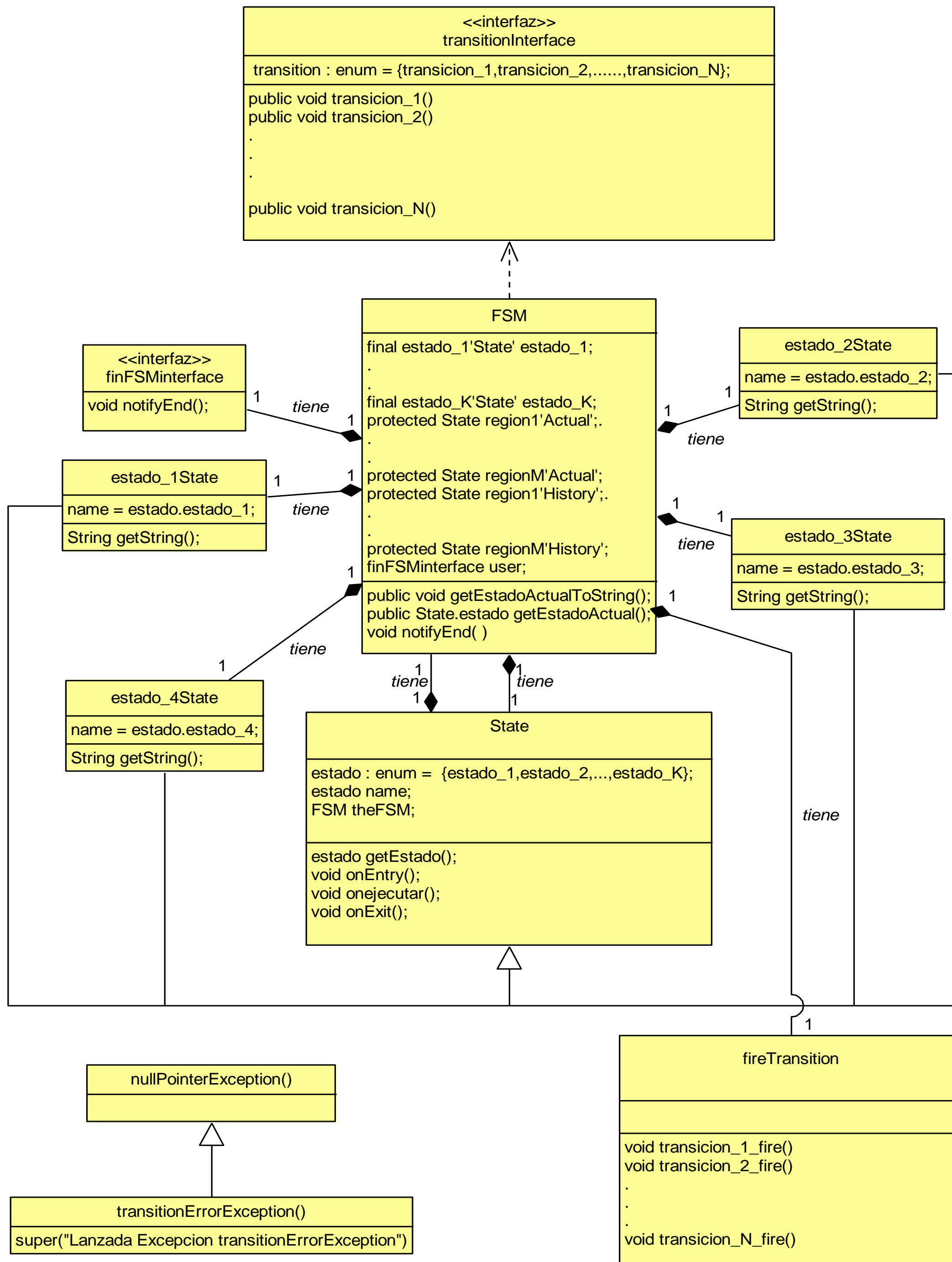


Figura 17. Diagrama de Clases.

Una vez visto el diagrama de clases que genera esta transformación, se muestra a continuación una breve explicación con la funcionalidad de cada fichero generado:

- ***interface finFSMinterface:*** Interfaz que incluye un método a través del cuál se notifica al simulador que debe finalizar.
- ***interface transitionInterface:*** Interfaz que indicara las posibles transiciones y las funciones que se llevarán a cabo al llegar una transición.
- ***clase transitionErrorException:*** Tipo de excepción que hereda de nullPointerException. Este tipo de excepción se lanza como respuesta a un intento de lanzamiento de transición no permitido, como puede ser el intento de una transición sin que la máquina de estados se encuentre en el estado fuente de dicha transición. Para permitir que este tipo de excepción pueda ser lanzada, a la hora de crear el modelo de ejemplo se debe establecer el valor throwException en el atributo unexpectedTransPolicy.
- ***clase FSM:*** Clase que implementa la interfaz transitionInterface, y se encarga de manejar la lógica de la máquina de estados, es decir, se encarga de recibir las llamadas a función por parte del simulador, que corresponderán con los deseos del usuario, y actuar en consecuencia. Para poder realizar las transiciones, esta clase contiene, mediante composición, un objeto de cada uno de los estados posibles y objetos de tipo State en los que se almacenan el estado actual de la máquina de estados y el estado actual de los pseudoestados de tipo histórico. Cuando llegue una transición, la clase FSM evalúa el estado actual de la máquina de estados para comprobar si esta transición puede ser ejecutada o no, en caso negativo, se muestra por pantalla esta situación y según el valor de unexpectedTransPolicy, la excepción será lanzada o no. Si por el contrario, la transición puede realizarse, se realizan las llamadas referentes a las acciones a realizar en los distintos estados involucrados en la transición y se simula el disparo utilizando un objeto del tipo fireTransition. Además de métodos para tratar las transiciones, dispone de un método que devuelve el estado actual de la máquina de estados en formato String y un método que devuelve el estado actual de la máquina de estados, como un tipo enumerado de datos. El tipo de datos enumerado ofrece mucha más funcionalidad al usuario que una un tipo de datos String. Por último, la clase FSM también se compone de un objeto de tipo finFSMinterface, el cual se utiliza en el método notifyEnd ( ). Mediante este método se notifica al simulador que debe finalizar, como consecuencia, por ejemplo, de la entrada a un estado de tipo final.
- ***clase state:*** Clase de la que heredarán los estados. Indicará los posibles estados y las funciones que se podrán realizar en ellos, como puede ser entrar, salir, permanecer en un estado…. Contiene un objeto de la clase FSM, este objeto será utilizado para actualizar las variables que guardan los estados actuales e históricos de la máquina de estados en la clase FSM. Este objeto se utiliza también en los estados de tipo final, para notificar a FSM la entrada a este tipo de estado, por lo que se notifica este hecho mediante su método notifyEnd ().
- ***Clase por cada estado:*** Se genera una clase por cada estado posible de la máquina de estados. Estos estados heredan de la clase State, dando el valor adecuado al atributo State, y realizando la implementación de los métodos asociados a cada estado. **El usuario debe introducir en estas clases, las acciones a realizar en cada uno de estos métodos**. Si el estado es de tipo final, no se establecen métodos asociados ni a la permanencia ni a la salida de este estado, utilizando el método notifyEnd () para notificar a FSM la entrada a este tipo de estado.

- ***clase fireTransition:*** Clase que implementará las funciones que simularán el disparo de una transición. El usuario puede introducir en esta clase el código asociado a cada disparo.
- ***clase Simulator:*** Simulador que se encarga de probar nuestra máquina de estados. Su funcionamiento será mostrar un menú a través de consola. Con este menú, el usuario elige la transición que desea ejecutar. Haciendo uso de composición, se añade a la clase Simulator, un objeto de la clase FSM. Simulador lanza las transiciones deseadas por el usuario, utilizando llamadas a funciones a través del objeto FSM. A la hora de crear el objeto de tipo FSM, se pasa a su constructor una referencia al propio simulador, creando así un mecanismo de callback, a través del cual la maquina de estados podrá comunicarse con el simulador. A través de este mecanismo de callBack, FSM invocará el método notifyEnd (), implementado en la clase Simulator y asociado a la interfaz finFSMinterface. La ejecución de este método tendrá como resultado el fin del simulador.

El código MOFSCRIPT que se utiliza para realizar este tipo de transformación, puede verse en el ANEXO B.1. El diagrama de secuencia se puede ver en la Figura 18, mostrada a continuación.

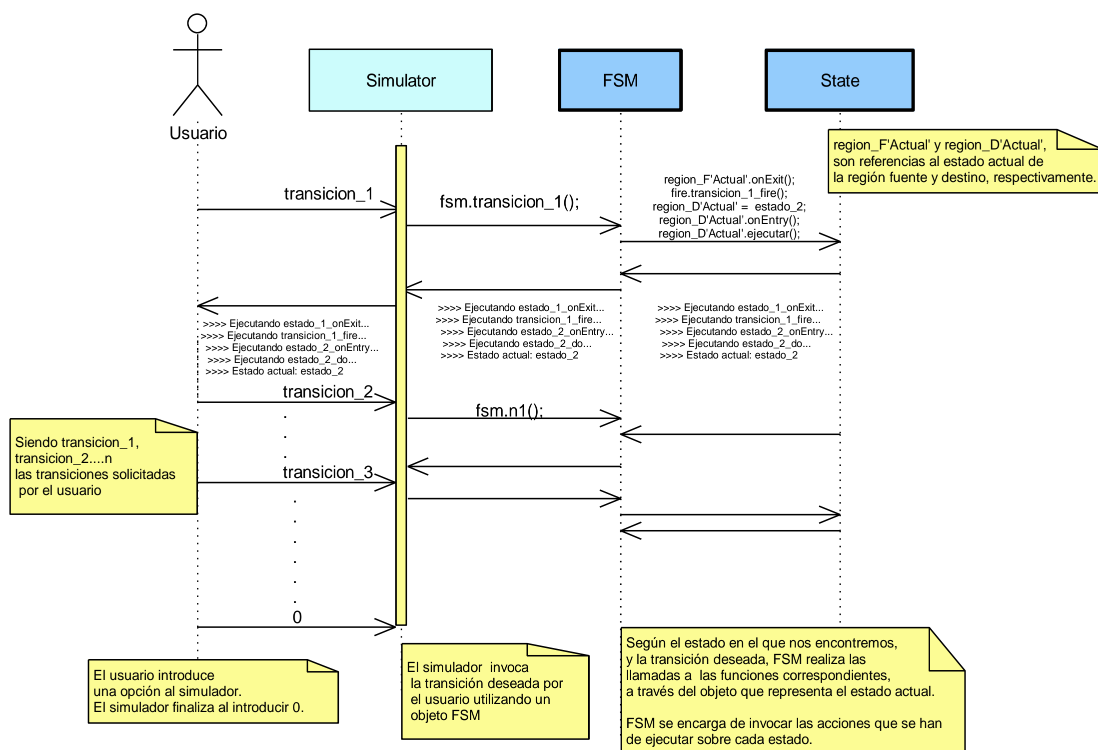


Figura 18. Diagrama de Secuencias

4.2.2 Transiciones a Funcion utilizando orientación a objetos: Ts2FOO.

La implementación de esta nueva transformación será muy parecida a la anterior, ya que como se ha comentado anteriormente, también se utiliza una orientación a objetos. En este caso se utilizará una única función para realizar las transiciones. Esta función recibe como parámetro un tipo enumerado de datos, con el que se indicará la transición a realizar. Para controlar este tipo de transiciones, se hará uso de un “switch global”, el cúal invoca un

método u otro en función del valor de este enumerado y el estado actual de la máquina de estados. El diagrama de clases que se muestra a continuación, en comparación con el visto en la Figura 17, varía sólo en la interfaz transitionInterface.

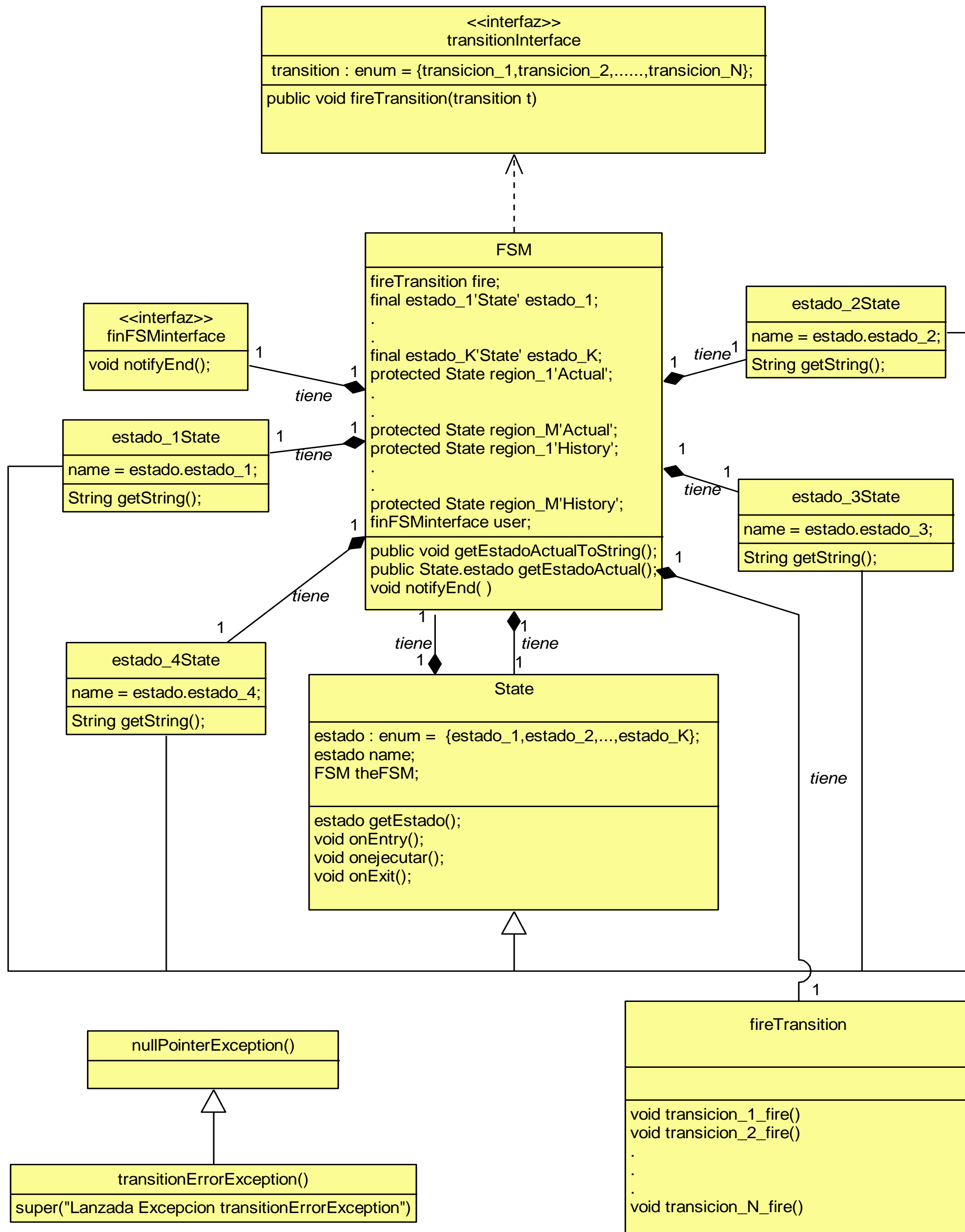


Figura 19. Diagrama de Clases.

La función desarrollada por cada una de estas clases e interfaces, se explica a continuación:

- ***interface finFSMinterface:*** Interfaz que incluye un método a través del cuál se notifica al simulador que debe finalizar.

- ***interface transitionInterface:*** Interfaz que indicara las posibles transiciones y las funciones que se llevarán a cabo al llegar una transición. En este caso, todas las transiciones serán invocadas a través de la misma función, indicándose en el parámetro de esta función el tipo de transición a realizar.
- ***clase transitionErrorException:*** Tipo de excepción que hereda de nullPointerException. Este tipo de excepción se lanza como respuesta a un intento de lanzamiento de transición no permitido, como puede ser el intento de una transición sin que la máquina de estados se encuentre en el estado fuente de dicha transición. Para permitir que este tipo de excepción pueda ser lanzada, a la hora de crear el modelo de ejemplo se debe establecer el valor throwException en el atributo unexpectedTransPolicy.
- ***clase FSM:*** Clase que implementará la interfaz transitionInterface, y se encarga de manejar la lógica de la máquina de estados, es decir, se encarga de recibir las llamadas a función por parte del simulador, que corresponderán con los deseos del usuario, y actuar en consecuencia. Para poder realizar las transiciones, esta clase contiene, mediante composición, un objeto de cada uno de los estados posibles y objetos de tipo State en los que se almacenan el estado actual de la máquina de estados y el estado actual de los pseudoestados de tipo histórico. Cuando se invoca el método fireTransition (transition t), que indica la llegada de una transición, la clase FSM comprueba mediante un switch global el valor de este enumerado, que indica la transición a realizar y después,mediante un segundo switch interior, evalúa el valor del estado actual de la máquina de estados y determina si la transición puede ser realizada o no. En caso negativo, se muestra por pantalla esta situación y según el valor de unexpectedTransPolicy, la excepción será lanzada o no. Si por el contrario, la transición puede realizarse, se realizan las llamadas referentes a las acciones a realizar en los distintos estados involucrados en la transición y se simula el disparo utilizando un objeto del tipo 'modeloName'_fireTransition. Además de métodos para tratar las transiciones, se ha implementado un método que devuelve el estado actual de la máquina de estados en formato String y un método que devuelve el estado actual de la máquina de estados, como un tipo enumerado de datos. El tipo de datos enumerado ofrece mucha más funcionalidad al usuario que una un tipo de datos String. Por último, la clase FSM también se compone de un objeto de tipo finFSMinterface, el cual se utiliza en el método notifyEnd ( ). Mediante este método se notifica al simulador que debe finalizar, como consecuencia, por ejemplo, de la entrada a un estado de tipo final.
- ***clase state:*** Clase de la que heredarán los estados. Indicará los posibles estados y las funciones que se podrán realizar en ellos, como puede ser entrar, salir, permanecer en un estado…. Contiene un objeto de la clase FSM, este objeto será utilizado para actualizar las variables que guardan los estados actuales e históricos de la máquina de estados en la clase FSM. Este objeto se utiliza también en los estados de tipo final, para notificar a FSM la entrada a este tipo de estado, por lo que se notifica este hecho mediante su método notifyEnd ().
- ***clase por cada estado:*** Se genera una clase por cada estado posible de la máquina de estados. Estos estados heredan de la clase State, dando el valor adecuado al atributo State, y realizando la implementación de los métodos asociados a cada estado. **El usuario debe introducir en estas clases, las acciones a realizar en cada uno de estos métodos**. Si el estado es de tipo final, no se establecen métodos

asociados ni a la permanencia ni a la salida de este estado, utilizando el método notifyEnd () para notificar a FSM la entrada a este tipo de estado.

- ***clase fireTransition:*** Clase que implementará las funciones que simularán el disparo de una transición. El usuario puede introducir en esta clase el código asociado a cada disparo.
- ***clase Simulator:*** Simulador que se encarga de probar nuestra máquina de estados. Su funcionamiento será mostrar un menú a través de consola. Con este menú, el usuario elige la transición que desea ejecutar. Haciendo uso de composición, se añade a la clase Simulator, un objeto de la clase FSM. Simulador lanza las transiciones deseadas por el usuario, utilizando la función fireTransition (transición t) a través del objeto FSM, el simulador indica la transición deseanda en el parámetro de esta función. A la hora de crear el objeto de tipo FSM, se pasa a su constructor una referencia al propio simulador, creando así un mecanismo de callback, a través del cual la maquina de estados podrá comunicarse con el simulador. A través de este mecanismo de callBack, FSM invocará el método notifyEnd (), implementado en la clase Simulator y asociado a la interfaz finFSMinterface. La ejecución de este método tendrá como resultado el fin del simulador.

Si se desea profundizar aún más en esta transformación, se puede encontrar su código MOFSCRIPT en el ANEXO B.2. El diagrama de secuencia se puede ver en la Figura 20, mostrada a continuación

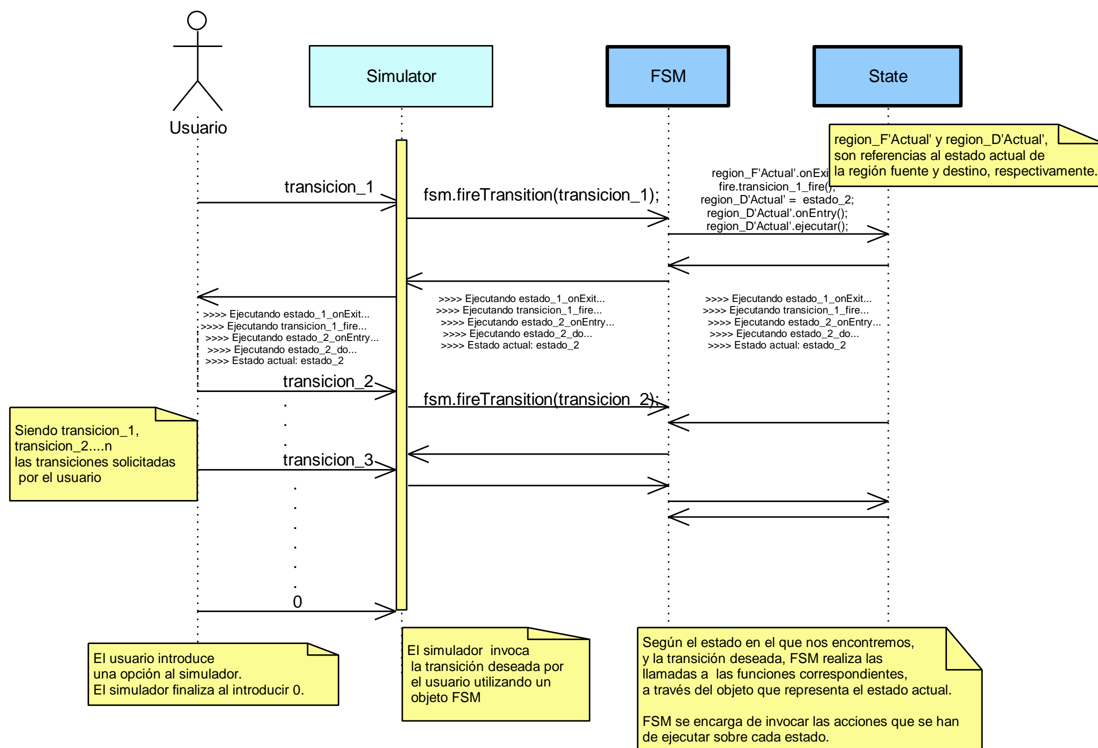


Figura 20. Diagrama de Secuencia

4.2.3 Patrón Estado

Implementación del patrón de diseño, "patrón estado". Siguiendo como referencia [**12**], se creará una clase por cada posible estado, todas estas clases heredarán de una clase "State" que tendrá las características comunes a todos los estados. Por otro lado, existe una clase que se encargará de invocar las funciones asociadas a cada transición, dependiendo del estado actual en el que se encuentre. Este estado actual estará almacenado en un objeto de tipo `State` que se añadirá mediante composición a esta última clase.

En la **Figura 21** Figura 21**Figura 21** se puede ver el diagrama de clases obtenido:

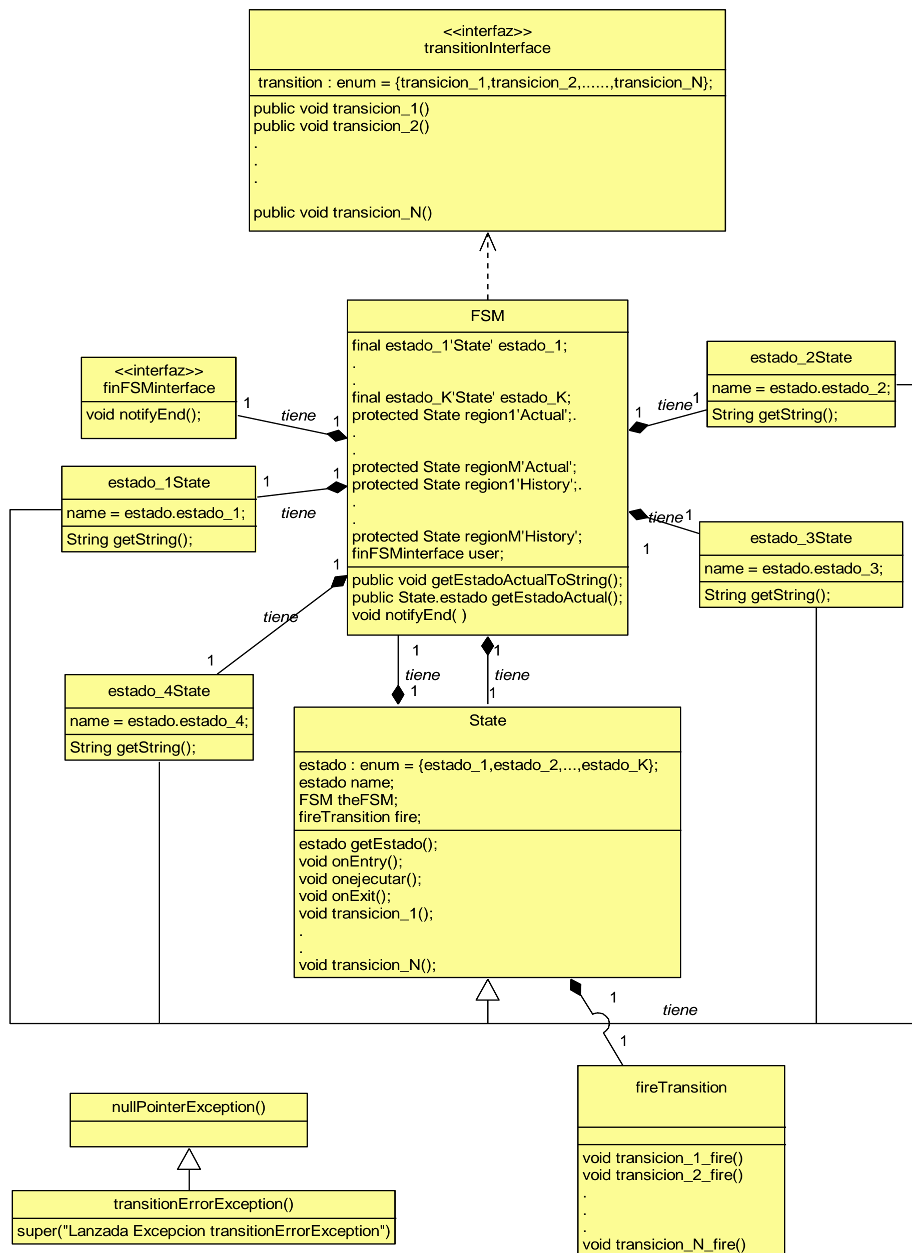


Figura 21. Diagrama de clases

Una vez visto el diagrama de clases resultante, se da una breve explicación sobre la implementación de cada clase:

- **Interface transitionInterface**: Interfaz que indicara las posibles transiciones y las funciones que se llevarán a cabo al llegar una transición.
- **Interface finFSMinterface**: Interfaz que incluye un método a través del cuál se informará el simulador que debe finalizar.
- **Clase fireTransition:** Clase que implementará las funciones que simularán el disparo de una transición. El usuario puede introducir en esta clase el código asociado a cada disparo.
- **Clase State**: Clase de la que heredarán los estados. Indicará los posibles estados y las funciones que se podrán realizar en ellos.Contiene un objeto de la clase FSM, este objeto será utilizado para actualizar las variables que guardan los estados actuales en FSM y en el caso de que se trate de un estado de tipo final, se informará a la máquina de estados de la entrada a este estado mediante su método notifyEnd().Mediante composición, esta clase tendrá un objeto del tipo fireTransition, el cuál utilizarán los estados para simular el disparo de una transición.
- **Clase por cada estado**: Una clase asociada a cada uno de los posibles estados. Heredan de la clase 'State', dando un valor al atributo 'name' ('estado_1',….,'estado_K'), según corresponda e implementando sus funciones, que corresponderán al comportamiento del estado. El usuario introducirá en esta clase, el código asociado a las acciones de entrada, salida y permanencia en un estado.

  Cada estado implementará sólo las transiciones que sea capaz de lanzar. Si un estado es de tipo final, al entrar en su método onEntry (), notificará la entrada a este tipo de estado a la FSM, mediante el método notifyEnd().
- **Clase FSM**: Clase que implementará la interfaz transitionInterface, se le asociarán, mediante composición, objetos que representen a cada uno de los posibles estados de la máquina de estados y un objeto de tipo State. Además de estos objetos, también se le asociará un objeto de tipo finFSMinterface, a través del cuál la máquina de estados notifica al simulador que debe finalizar, como consecuencia por ejemplo de la entrada en un estado de tipo final. Esta clase será la encargada de comprobar si una transición se puede ejecutar o no, invocando los métodos necesarios en caso afirmativo. El valor inicial de State será el de un objeto de la clase InitState.
- **Clase Simulator:** Simulador que se encargará de probar nuestra máquina de estados. Su funcionamiento será mostrar un menú a través de consola. Con este menú, el usuario podrá elegir la transición que desea ejecutar. Haciendo uso de composición, se añadirá a la clase Simulator, un objeto de la clase FSM. Simulador lanzará las transiciones deseadas por el usuario, utilizando llamadas a funciones del objeto FSM. A la hora de crear el objeto de tipo FSM, se pasa a su constructor una referencia al propio simulador, simulando así un mecanismo de callback, a través del cual la maquina de estados podrá comunicarse con el simulador. Simulador implementará el método notifyEnd (), asociado a la interfaz finFSMinterface, la ejecución de este método tendrá como resultado el fin del simulador.

A continuación, en la Figura 22, se puede ver el diagrama de secuencia para una máquina de estados jerárquica.

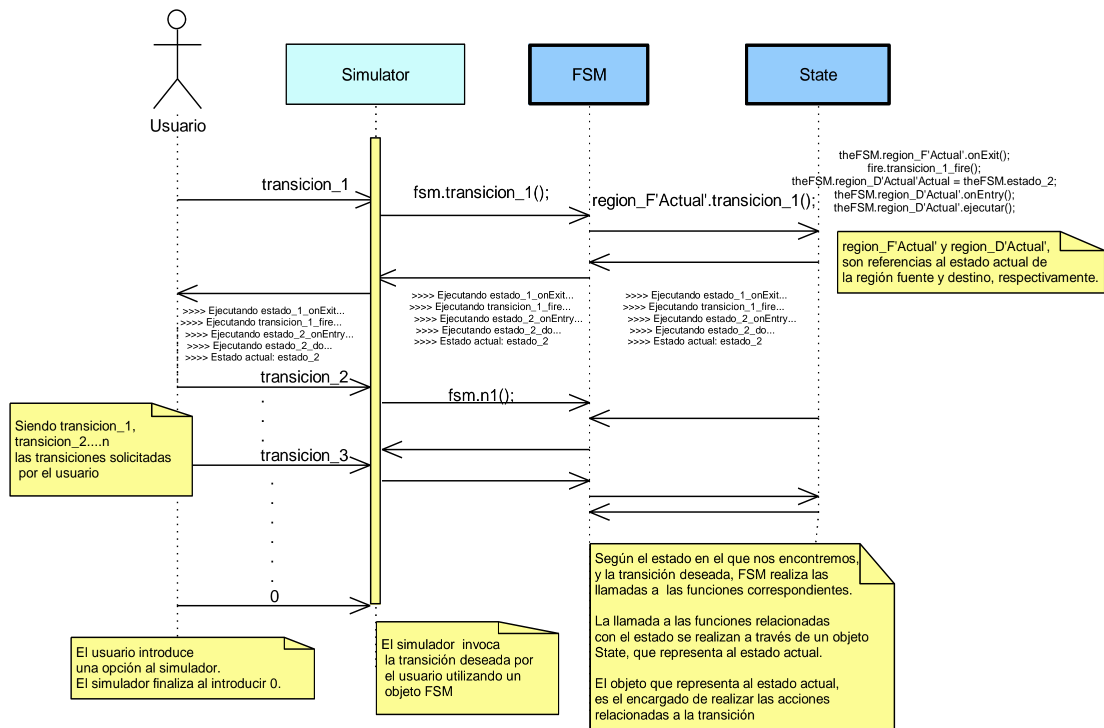


Figura 22. Diagrama de Secuencia

El código MOFSCRIPT necesario para realizar este tipo de transformación, puede encontrarse en el ANEXO B.3.

4.3 MODELOS DE EJEMPLO.

Para poder comprobar el correcto funcionamiento de las transformaciones se deben crear modelos a partir del metamodelo de máquina de estados jerárquicas, teniendo en cuenta las características de este tipo de metamodelo. Para una mejor comprensión, se irán proponiendo varios ejemplos, en los que se irán aumentando gradualmente las características recogidas. El código JAVA generado a partir de estos modelos, puede verse en el CD que acompaña este trabajo.

PRIMER MODELO.

En este primer modelo se comprueban las siguientes características:

- Máquina de estados formada por estados y transiciones entre estados dentro de una región.
- Sólo presenta un pseudoEstado, es de tipo Inicial.
- Tratamiento de estados de tipo final.

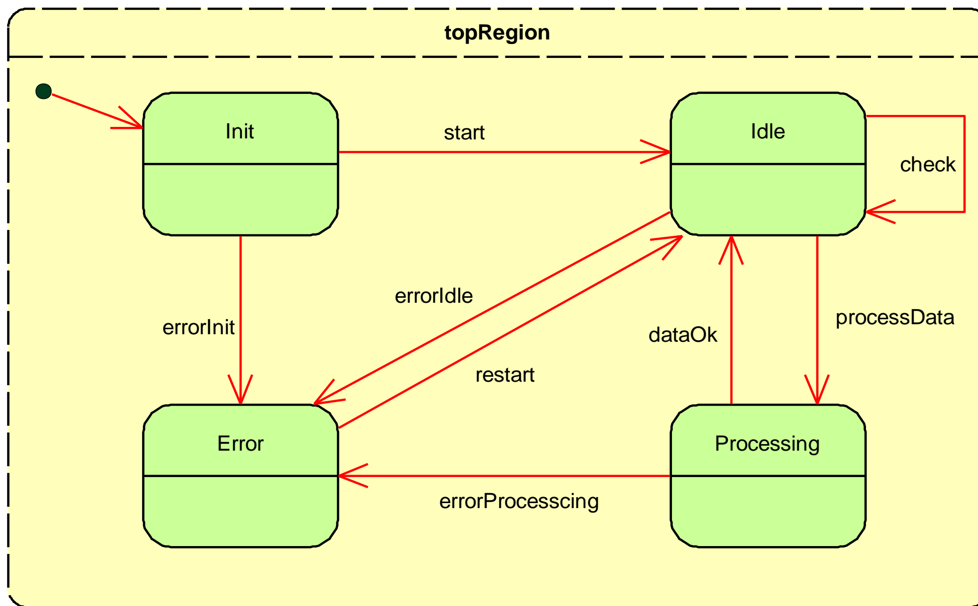


Figura 23. Modelo 1. Soporte para regiones

SEGUNDO MODELO.

En este segundo modelo, además de la existencia de regiones se añade la posibilidad de realizar transiciones entre las distintas regiones, en este caso, antes de ejecutar una transición desde un estado que contiene transiciones, se deben abandonar los estados de las regiones interiores. Se debe tener en cuenta que a la hora de ejecutar una transición hacia un estado que contiene regiones interiores, se entrará por defecto a los estados apuntados por los pseudoestados inicial o históricos (sólo puede existir uno) dentro de cada región interior al estado.

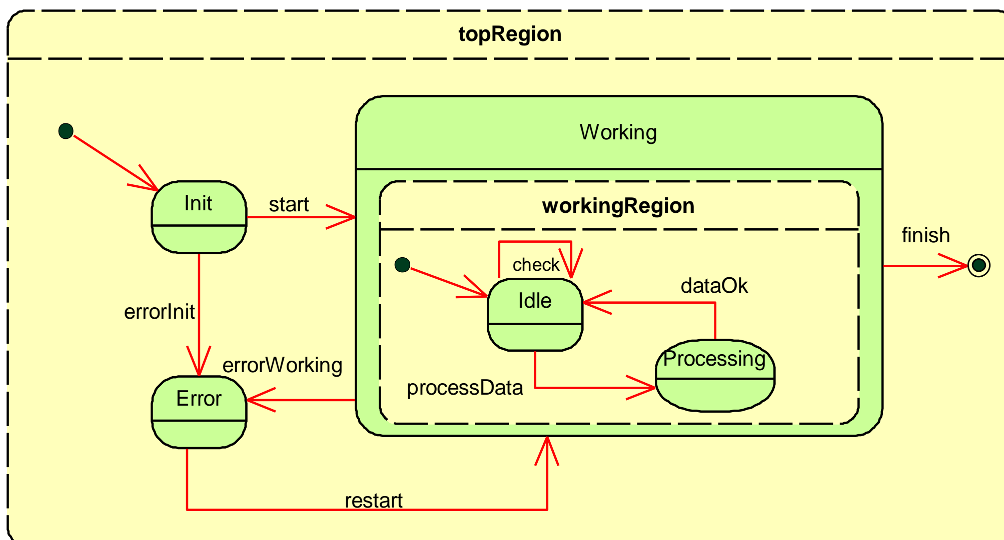


Figura 24. Modelo II: Transiciones entre distintas regiones

TERCER MODELO.

En este último ejemplo se recogen todas las características del metamodelo, es decir, existencia de regiones, transiciones entre regiones y todo tipo de pseudoestados. En el anexo B.4 se puede ver un ejemplo de simulación del modelo mostrado en la Figura 25. Las nuevas características recogidas en este ejemplo, respecto a los dos ejemplos anteriores, son la existencia de pseudoestados de tipo fork y join.

- Join: La transición desde un pseudoEstado de tipo join, sólo se ejecutará si la máquina de estados se encuentra en unos estados determinados.
- Fork: La transición se divide en varias transiciones, es decir, se pasará de un único estado fuente a varios destinos. Se debe tener en cuenta, que estos destinos estarán ubicados en distintas regiones de un mismo estado.

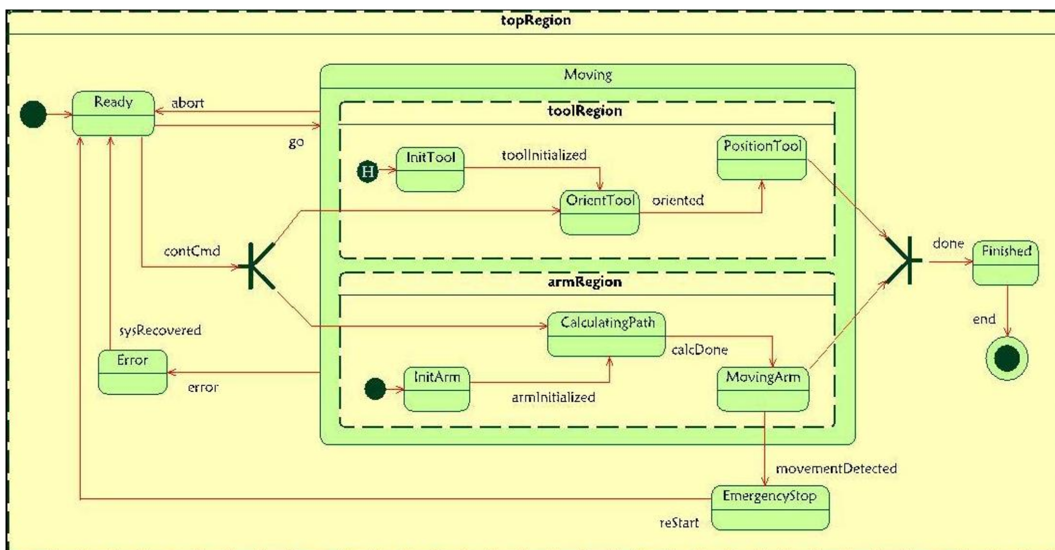


Figura 25. Modelo III: Inclusión de pseudoestados Fork y Join

5. CONCLUSIONES Y TRABAJOS FUTUROS.

La realización de este Trabajo Fin de Grado se ha dividido en dos fases. La primera fase de ellas ha consistido en el estudio de las máquinas de estado finito y redes de Petri, analizando los avances ya conseguidos en esta materia, desde las máquinas de Moore y Mealy, pasando por la ampliación de máquinas de estados que realiza David Harel (stateCharts),el desarrollo de software dirigido por modelos y finalizando con el estándar UML 2, que permite el modelado. El trabajo está basado en los avances en el desarrollo de software dirigido por modelos en los que cada concepto debe ser modelado. Las transformaciones modelo a texto se han realizado gracias a la posibilidad de crear modelos que representen el funcionamiento de un sistema y con la ayuda de las herramientas adecuadas, en este trabajo se ha utilizado la herramienta MOFSCRIPT.

De esta forma se ha conseguido transformar el modelo del sistema, que aunque sea una forma bastante buena de describir el funcionamiento del sistema, no deja de ser un gráfico, a una generación de texto que permite la implementación del funcionamiento del sistema. Particularmente, en este Trabajo Fin de Grado se ha realizado la transformación a código JAVA, creando las clases e interfaces necesarias, entre las que se puede encontrar un simulador que permite realizar una prueba del funcionamiento del sistema modelado y posteriormente transformado.

La segunda fase ha consistido en del desarrollo de código MOFSCRIPT capaz de generar código JAVA a partir de un modelo, realizando así las transformaciones modelo a texto. Este modelo debe seguir el patrón impuesto por un metamodelo. En concreto se ha utilizado un tipo de metamodelo que recoge las características de las máquinas de estado jerárquicas.

Se realizan tres transformaciones sobre este metamodelo. Estas transformaciones tienen los siguientes objetivos:

- Ts2Fs: Cada transición del modelo será implementada mediante una función.
- Ts2F: Todas las transiciones del modelo utilizarán la misma función, a la hora de ser invocadas, para distinguir entre las distintas transiciones, se pasa como parámetro de esta función la transición a realizar.
- Patrón Estado: Sigue los pasos recomendados por el patrón de diseño patrón estado.

A la hora de utilizar las transformaciones implementadas para sistemas que simulan el comportamiento de máquinas de estados jerárquicas, se deben tener en cuenta las siguientes restricciones:

- El sistema estará contenido en una región global, la cual se denomina topRegion.
- No deben existir estados ni transiciones con el mismo nombre.
- Cada región debe tener un único pseudoEstado inicial o histórico, no pudiendo tener los dos pseudoestados a la vez.
- El sistema no puede estar en varios estados a la vez, aunque si puede estar en varias regiones de un mismo estado a la vez.
- Debido a la necesidad de aumentar la complejidad en el código, las transiciones que tienen como fuente un pseudoEstado de tipo fork o las transiciones que tienen como destino un pseudoEstado de tipo Join, sólo deben atravesar un nivel de región. Resaltar que esta restricción en cuanto a transiciónes entre regiones sólo se refiere a transiciones cuya fuente sea un pseudoEstado de tipo fork o cuyo destino sea un pseudoEstado de tipo Join.

Este tipo de transformaciones están recomendadas para sistemas complejos en los que se establezca una jerarquía entre estados. Soportando, además de la existencia de regiones, la utilización de pseudoestados de tipo inicial, histórico, join y fork.

Debido a que estas transformaciones sólo proporcionan una implementación de la lógica de la máquina de estados, una vez se ha seleccionado la transformación más conveniente para el modelo realizado y se ha ejecutado, el usuario debe rellenar los espacios reservados en el código con la implementación que defina el comportamiento específico de la máquina de estados, como puede ser las acciones a realizar al entrar, salir o mientras permanezca en un estado, las acciones a realizar cuando se produzca el disparo de una transición, etc.

Como trabajos futuros se propone la tarea de solventar el problema que supone la restricción impuesta en las transiciones que tienen como fuente un pseudoEstado de tipo fork o como destino un pseudoEstado de tipo Join, eliminando esta restricción por completo.

Una vez superada esta restricción sería un gran avance realizar estas transformaciones modelo a texto, de manera que se genere código en otros lenguajes de programación.

Otra posible tarea que se puede llevar a cabo es mediante la utilización de OCL, especificar restricciones, que aseguren que los modelos de máquinas de estados son correctos, por ejemplo se debe cumplir que cada región contenga un pseudoEstado de tipo inicial o histórico (sólo uno), ya que sólo con el meta-modelo no es posible especificar todas las restricciones.

6. ANEXOS

Este capítulo contiene uns serie de anexos entre los que se incluye un pequeño manual que explica los pasos seguidos para realizar las transformaciones modelo a texto, desde la ceración del fichero MOFSCRIPT hasta su ejecución sobre el modelo. Además de este manual, se adjunta el código MOFSCRIPT implementado para realizar cada una de las transformaciones:

ANEXO A: TRANSFORMACIÓN M2T UTILIZANDO MOFSCRIPT.

Para realizar la transformación de lenguaje, modelo a texto, se hará uso de la herramienta MOFScript, incorporada en Eclipse como plug-in.

Para poder utilizar MOFSCRIPT, se debe comprobar que se encuentra instalado como plug-in de eclipse, para ello, navegando por el menú superior de Eclipse, se pueden observar los plug-in que se encuentran instalados, ver Figura 28.

En esta figura observa que los plug-in de MOFScript se encuentran instalados. Una vez comprobado que MOFSCRIPT se encuentra instalado, se debe realizar la configuración de MOFSCRIPT mediante el menú de preferencias: "Window => Preferences". Ver Figura 29.

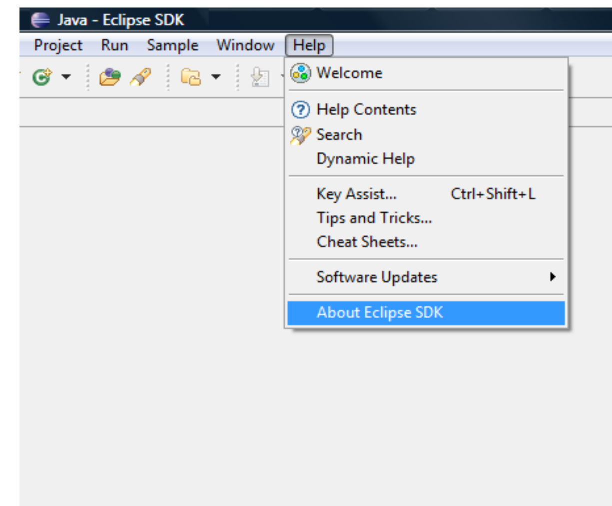


Figura 26. Acceso Información sobre Eclipse.

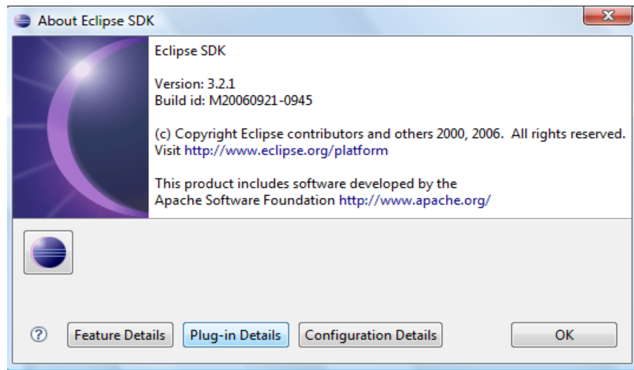
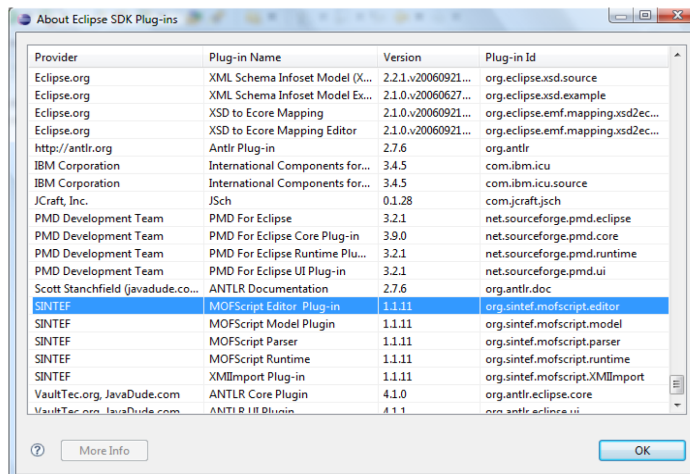


Figura 27. Acceso plugins instalados



About Eclipse SDK Plug-ins

Provider	Plug-in Name	Version	Plug-in Id
Eclipse.org	XML Schema Infoset Model (X...	2.2.1.v20060921...	org.eclipse.xsd.source
Eclipse.org	XML Schema Infoset Model Ex...	2.1.0.v20060627...	org.eclipse.xsd.example
Eclipse.org	XSD to Ecore Mapping	2.1.0.v20060921...	org.eclipse.emf.mapping.xsd2ec...
Eclipse.org	XSD to Ecore Mapping Editor	2.1.0.v20060921...	org.eclipse.emf.mapping.xsd2ec...
http://antlr.org	Antlr Plug-in	2.7.6	org.antlr
IBM Corporation	International Components for...	3.4.5	com.ibm.icu
IBM Corporation	International Components for...	3.4.5	com.ibm.icu.source
JCraft, Inc.	JSch	0.1.28	com.jcraft.jsch
PMD Development Team	PMD For Eclipse	3.2.1	net.sourceforge.pmd.eclipse
PMD Development Team	PMD For Eclipse Core Plug-in	3.9.0	net.sourceforge.pmd.core
PMD Development Team	PMD For Eclipse Runtime Plu...	3.2.1	net.sourceforge.pmd.runtime
PMD Development Team	PMD For Eclipse UI Plug-in	3.2.1	net.sourceforge.pmd.ui
Scott Stanchfield (javadude.co...	ANTLR Documentation	2.7.6	org.antlr.doc
SINTEF	MOFScript Editor Plug-in	1.1.11	org.sintef.mofscript.editor
SINTEF	MOFScript Model Plugin	1.1.11	org.sintef.mofscript.model
SINTEF	MOFScript Parser	1.1.11	org.sintef.mofscript.parser
SINTEF	MOFScript Runtime	1.1.11	org.sintef.mofscript.runtime
SINTEF	XMIImport Plug-in	1.1.11	org.sintef.mofscript.XMIImport
VaultTec.org, JavaDude.com	ANTLR Core Plugin	4.1.0	org.antlr.eclipse.core

More Info   OK

Figura 28. Plugins instalados

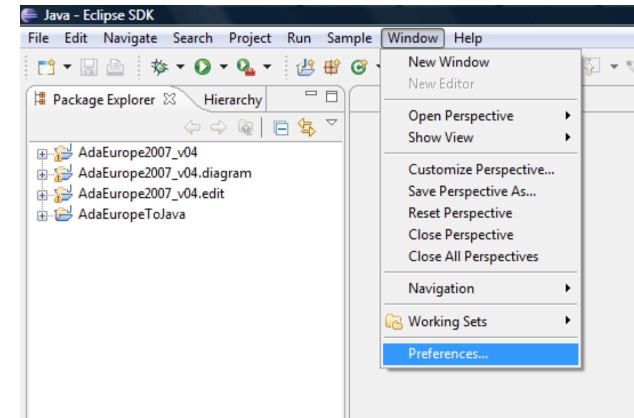


Figura 29. Acceso a Menú de Preferencias

Debe aparecer una ventana parecida a la mostrada en la Figura 30, en la que se puede acceder al menú de preferencias de MOFSCRIPT seleccionando en el menú de la izquierda la opción "MOFScript Preferences" para proceder a su configuración. Se debe revisar que el valor de los campos mostrados en la Figura 30, es el deseado. El significado de estos campos es el siguiente:

- **Metamodel path:** Indica la dirección de la cual la herramienta MOFScript recoge los metamodelos para la transformación.
- **Model path:** Dirección por defecto que utiliza el editor para buscar modelos de entrada.
- **Transformation path:** Actualmente no usado.
- **Root generation directory:** Directorio donde se almacenarán los archivos de salida(generados por la herramienta).

- **Prefix for project generation:** Prefijo dado a los proyectos generados por Eclipse (su valor por defecto es “mofscript-gen”).
- **Generate project:** Valor booleano que determina si un proyecto Eclipse debe ser generado, o no.
- **Import path:** Dirección utilizada para el análisis de la transformación. Debe ser una lista de directorios separados por “;”
- **Traceability active:** Activa o desactiva el seguimiento. Actualmente, la implementación del seguimiento no está finalizada. Activandolo ahora, sólo imprimirá algunos mensajes en pantalla.
- **Trace model generation dir**: Dirección de almacenamiento del seguimiento.

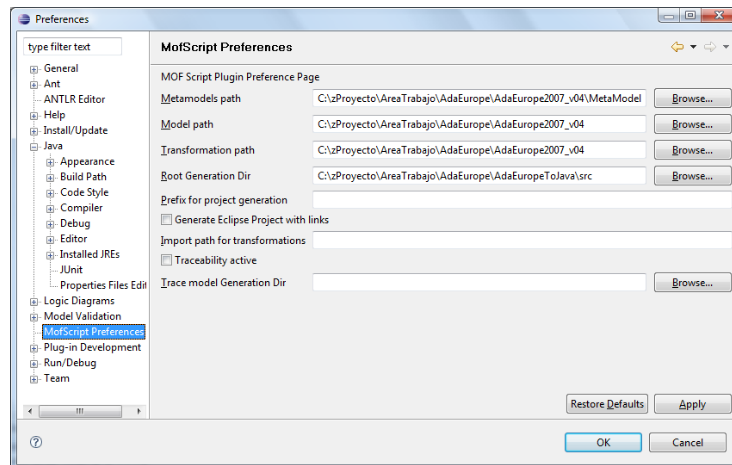


Figura 30. Menú de Preferencias de MOFSCRIPT

Si se desea realizar una transformación a código JAVA, sería recomendable crear antes una nueva carpeta JAVA e indicar que los archivos generados se almacenen en dicha carpeta, así los archivos “.java” serán compilados, nada más ser creados.

Una vez rellenados estos campos, es recomendable revisar que Eclipse esté utilizando la versión de compilador de Java correcta, la que se encuentre instalada en el ordenador. Esta comprobación se puede realizar accediendo al menú de la izquierda del menú de preferencias, seleccionando la opción “Java => Compiler”, ver Figura 31.

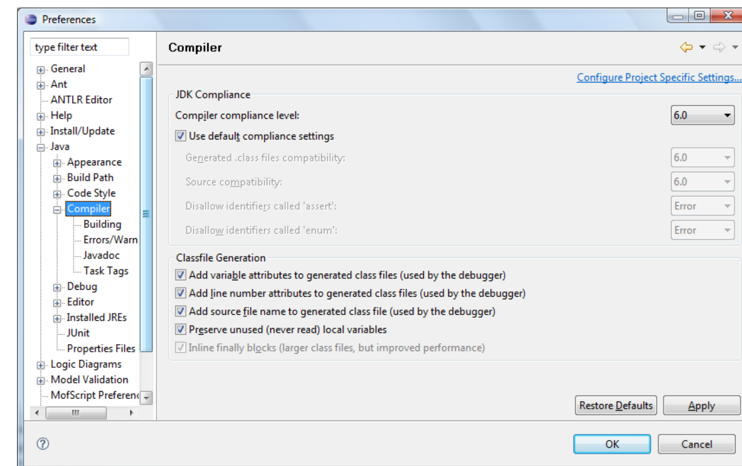
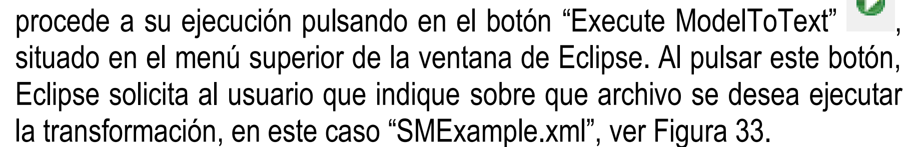


Figura 31. Compilador JAVA

Una vez realizada la configuración de la herramienta, se puede comenzar a realizar la transformación modelo a texto. Para ello, se debe crear un nuevo editor MOFScript seleccionando “File => New => Other” y eligiendo la opción “MOFSCRIPT File”, como se puede ver en la Figura 32.Se generará

un archivo con extensión .m2t, en el que se implementará la transformación.

Una vez implementado el código y comprobado que no hay errores, Eclipse se encarga de compilar el código cada vez que se guarda el archivo, se procede a su ejecución pulsando en el botón “Execute ModelToText” , situado en el menú superior de la ventana de Eclipse. Al pulsar este botón, Eclipse solicita al usuario que indique sobre que archivo se desea ejecutar la transformación, en este caso “SMExample.xml”, ver Figura 33.

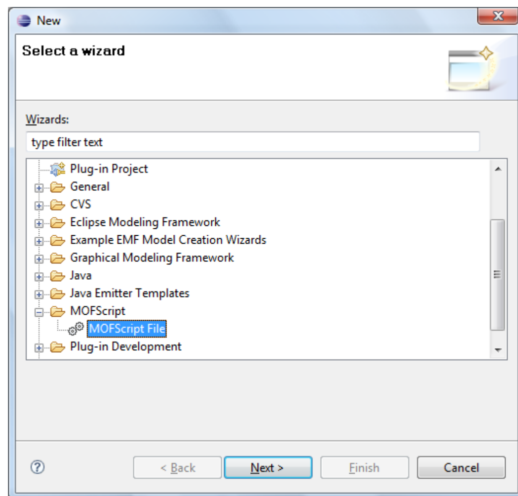


Figura 32. Creación Fichero MOFScript

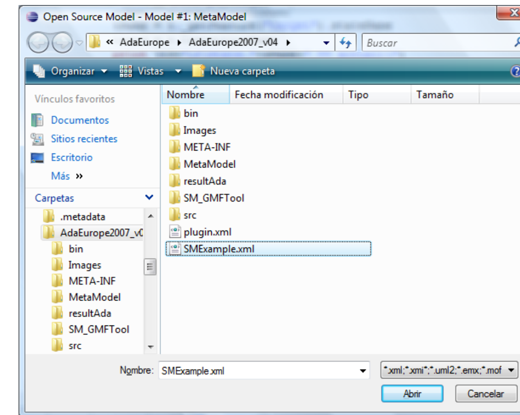


Figura 33. Selección del modelo a transformar

Una vez ejecutado, se generarán los archivos correspondientes a la transformación escogida en el directorio que se haya indicado en la configuración de MOFScript.

ANEXO B: CÓDIGO MOFSCRIPT TRANSF. M2T.

B. 1 Código MOFSCRIPT. Ts2FsOO.

```
/**
 * transformation SM_ModeltoJava_Ts2FsOO
 */
texttransformation Ts2FOO (in MetaModel:StateMLplus) {

        /*Se declaran estas variables globales, porque se usarán tanto en el main,
        como en los métodos utilizados fuera de main*/

        var exceptionName : String = "transitionErrorException" //Nombre de la
                              excepcion creada para transiciones erroneas

        var fatherException :String = "NullPointerException" //Nombre de la excepcion de
                              JAVA de la cual hereda

        var regionSM : String = self.topRegion.name

        var transPolicy : String = self.unexpectedTransPolicy

    MetaModel.StateMachine::main(){

        var path : String = "StateMLplus/"+self.name+"/Ts2FsOO"

        var package : String = "StateMLplus."+self.name+".Ts2FsOO"

        var states : List = getStates ()

        var transitions : List = getTransitions()

        var transitionNames:List = getTransitionNames ( )

        var regiones : List = self.objectsOfType(MetaModel.Region)

        var historyPseudo : List = self.objectsOfType(MetaModel.Pseudostate)->
        select(ph:MetaModel.Pseudostate | (ph.kind =="HistoryState"))

        //---------------------------------------------

        //-------CLASE transitionErrorException--------

        //---------------------------------------------

        /*Esta clase sólo se creará si a la hora de crear el modelo, se elige la
        opción UnexpectedTransPolicies => throwException*/

        if(transPolicy = "throwException")

        {

                file excep(path+"\\"+exceptionName+".java");

                excep.println("package "+package+";");

                excep.println("public class "+exceptionName+ " extends
                "+fatherException+"{");

                excep.println("\t public "+exceptionName+"()\n \t {");

                excep.println('\t\t super("Lanzada excepción
                '+exceptionName+'");');

                excep.println("\t }");

                excep.println("}");

        }


        //---------------------------------------------

        //-----------interfaz finFSM------------------

        //---------------------------------------------

        /*Esta interfaz será implementada por FSM, e incluirá el método mediante el
        cuál un estado final notificará que se ha accedido a un estado final.Esta proceso
        se llama proceso de CALLBACK*/

        file finInterf(path+"\\"+self.name+"_finFSMinterface.java");

        finInterf.println("package "+package+";");

        finInterf.println("public interface "+self.name+ "_finFSMinterface{");

        finInterf.println("\t public void notifyEnd();");

        finInterf.println("}");

        //---------------------------------------------

        //------INTERFAZ transitioInterface-----------

        //---------------------------------------------

        file trs(path+"\\"+self.name+"_transitionInterface.java")

        trs.println("package "+package+";");

        trs.println("public interface "+self.name +"_transitionInterface{");

        trs.print("\t public enum transition { ")

                transitionNames -> forEach ( t : String ) {

                        if (position()!=0) trs.print (", ")

                        trs.print ( t )

                }

        trs.println("};\n")

        transitionNames->forEach(t:String)

        {
```

```
        trs.println("\t public void "+t+"();")
    }
    trs.println("}"); //FIN INTERFACE TRANSITION
    //--------------------------------------------
    //---------------CLASE FIRE--------------------
    //--------------------------------------------
    file fire(path+"\\"+self.name+"_fireTransition.java")
    fire.println("package "+package+";");
    fire.println("class "+self.name +"_fireTransition{");
    fire.println("\n\t /*ESTA CLASE CONTIENE LAS IMPLEMENTACIONES DE LAS
    FUNCIONES");
    fire.println("\t DISPAROS DE LAS TRANSICIONES*/");
    fire.println("\n\t /*EL USUARIO INSERTARÁ EN ESTAS FUNCIONES,");
    fire.println("\t EL CÓDIGO DESEADO PARA CADA DISPARO*/");
    /*Funciones que simulan el disparo de las transiciones*/
    transitions->forEach(trans : MetaModel.Transition)
    {
        fire.println("\t void "+trans.name+"_fire(){");
        fire.println ('\t\t System.out.println (" \t\t>>>> Ejecutando
        '+trans.name+'_fire...");')
        fire.println("\t}");
    }
    fire.println("}");
    //--------------------------------------------
    //---------------CLASE FSM---------------------
    //--------------------------------------------
    file fsm(path+"\\"+self.name+"_FSM.java")
    fsm.println("package "+package+";");
    fsm.println("public class "+self.name+"_FSM implements
    "+self.name+"_transitionInterface {");
    /*Se utiliza un objeto de la clase fireTransition, para simular el   disparo     de
    las transiciones*/
    fsm.println("\n\t/*Se utiliza un objeto de la clase fire, para simular el
    disparo de las transiciones*/");
    fsm.println("\t "+self.name+"_fireTransition fire = new
    "+self.name+"_fireTransition();\n");
    /*SE CREA UN OBJETO POR CADA POSIBLE ESTADO*/
    fsm.println("\n\t /*Se crea un objeto por cada estado*/");
    states->forEach(s:MetaModel.State)
    {
        fsm.println("\t final "+self.name+"_"+s.name+"State "+s.name+";")
    }
    /*Ahora se evaluan los pseudoEstados, serán de tipo join*/
    states -> forEach (p : MetaModel.Pseudostate)
        fsm.println("\t final "+self.name+"_"+p.name+"State "+p.name+";")

    /*Se crean las variables que contendrán el estado actual de cada región*/
    regiones->forEach(reg:MetaModel.Region)
    {
        fsm.println("\t protected "+self.name+"_State
        "+reg.name+"Actual;");
    }
    /*Variables para las regiones que tengan historico*/
    historyPseudo->forEach(ph:MetaModel.Pseudostate)
    {
        fsm.print("\t protected "+self.name+"_State
        "+ph.owner.name+"History;")
    }
    fsm.println("\n\t /*Se crea objeto de tipo finFSMinterface*/");
    fsm.println("\t "+self.name+"_finFSMinterface user;")

    fsm.println("\n\t /*SE CREA EL CONSTRUCTOR*/")
    fsm.println("\n\t public "+self.name+"_FSM("+self.name+"_finFSMinterface
    userNot){")
    fsm.println("\n\t /*SE INICIALIZAN LOS ESTADOS*/")
    states->forEach(s:MetaModel.State)
    fsm.println("\t\t "+s.name+" = new "+self.name+"_"+s.name+"State(this);")
    states->forEach(p:MetaModel.Pseudostate)
```

```
fsm.println("\t\t "+p.name+" = new "+self.name+"_"+p.name+"State(this);")

fsm.println("\n\t /*Se inicializan topRegion y las regiones que tengan
historicos*/")

fsm.println("\t\t "+regiones.first().name+"Actual =
"+getPrimerEstadoName(regiones.first())+";")

/*Se crea una variable para cada region que tenga un pseudoEstado historico    */

historyPseudo->forEach(ph:MetaModel.Pseudostate)

{

          fsm.print("\t\t "+ph.owner.name+"History =
          "+getPrimerEstadoName(ph.owner)+";")

}

fsm.println("\n\t/*Se inicializa el objeto que notificará el fin de la
maquina de estados*/")

fsm.println("\t\t user = userNot;");

fsm.println("\n\t }")//fin del constructor


fsm.println("\n\t/*Se crea un tipo de datos enumerado que contendrá los
estados actuales de cada región*/");

fsm.println("\t public enum region {");

regiones->forEach(reg:MetaModel.Region)

{

  /*Solo  se  inicializa  topRegión,  ya  que  será  a  la  que  se  entra  nada  más
poner en funcionamiento la máquina de estados. Las demás variables se inicializan
a null*/

          if(position() ==0)

                    fsm.print("\t\t
          "+reg.name+"("+self.name+"_State.estado."
+getPrimerEstadoName(reg) +")")

          if(position() !=0)

          {

                    fsm.print(",\n")

                    fsm.print("\t\t "+reg.name+"(null)")

          }

}

fsm.println(";")

fsm.println("\n\t\t public "+self.name+"_State.estado
currentState;/*Variable que almacenará el estado actual*/")
```

```
 fsm.println("\n\t /*Se crea un constructor para inicializar la varible de
estado    actual, currentState*/")

fsm.println("\t\t region ("+self.name+"_State.estado estadoIni) {");

fsm.println("\t\t\t currentState = estadoIni;");

fsm.println("\t\t }");

fsm.println("\t }");

fsm.println("\n\t/*Se crea una instancia del enumerado region, para poder
acceder a sus campos*/");

fsm.println("\t/*Se debe inicializar a cualquier valor del enumerado, da
igual cual sea*/")

fsm.println("\t region estadoActual = region."+regiones.first().name+";")

/*FUNCION QUE DEVOLVERÁ EL ESTADO ACTUAL DE LA MÁQUINA DE ESTADOS.

DEVOLVERÁ  UN  TIPO  DE  DATOS  ENUMERADO,  QUE  CONTENDRÁ  EL  ESTADO  ACTUAL  DE
CADA REGIÓN*/

fsm.println("\n\t /*FUNCION QUE DEVOLVERÁ EL ESTADO ACTUAL DE LA MÁQUINA DE
ESTADOS.");

fsm.println("\t DEVOLVERÁ UN TIPO DE DATOS ENUMERADO, QUE CONTENDRÁ EL
ESTADO ACTUAL DE CADA REGIÓN*/");

fsm.println("\t public region getEstadoActual (){");

fsm.println("\t\t return estadoActual;")

fsm.println("\t }");

/*FUNCION QUE SE ENCARGARÁ DE ACTUALIZAR EL VALOR DEL ENUMERADO QUE  CONTIENE  LOS
ESTADOS ACTUALES*/

fsm.println("\n\t/*FUNCION QUE SE ENCARGARÁ DE ACTUALIZAR EL VALOR DEL
ENUMERADO QUE CONTIENE LOS ESTADOS ACTUALES*/");

fsm.println("\t public void actualizarEstados(){")

regiones->forEach(reg:MetaModel.Region)

{

          fsm.println("\t\t if("+reg.name+"Actual != null)");

          fsm.println("\t\t\t estadoActual."+reg.name+".currentState =
          "+reg.name+"Actual.name;")

          fsm.println("\t\t else ")

          fsm.println("\t\t\t estadoActual."+reg.name+".currentState =
          null;")

}

fsm.println("\t }")
```

```
/*FUNCION QUE DEVOLVERÁ LOS ESTADOS ACTUALES DE TODAS LAS REGIONES EN
FORMATO STRING*/

fsm.println("\n\t /*FUNCION QUE DEVOLVERÁ LOS ESTADOS ACTUALES DE TODAS LAS
REGIONES*/");

fsm.println("\t public void getEstadosActualesToString(){")

        regiones->forEach(reg:MetaModel.Region)

                {

                        fsm.println("\t\t\t\t if("+reg.name+"Actual
                        != null)");

                        fsm.println('\t\t\t\t\t
                         System.out.println(">>>>>>>>>>
                Region '+reg.name+' en estado:"
        +'+reg.name+ 'Actual.name);');

                        fsm.println("\t\t\t\t else");

                        fsm.println('\t\t\t\t\t
                        System.out.println(">>>>>>>>>>
                Region '+reg.name+' no activa");');

                }

fsm.println("\t }")

fsm.println ("\t // -------------------------------------");

//FUNCIONES PARA LAS TRANSICIONES

transitions -> forEach ( tn : MetaModel.Transition )

{

        println ("\t // -------------------------------------")

        print ("\t\t public void " + tn.name + "() throws
        NullPointerException ")

/*La siguiente excepción sólo se lanzará si se elige la opción
unexpectedTransPolicy => throwException*/

        if(transPolicy = "throwException")

                {

                        fsm.print(","+exceptionName)

                }

        fsm.println(" {")

        fsm.println("\t\t   try\n\t\t   {//Parte de código que puede
        provocar excepciones")

        writeProcedure ( tn )

        fsm.println("\t\t    }//Fin try")

                /*La siguiente excepcion sólo se lanzará si a la hora de crear el
                modelo, se elige la opción UnexpectedTransPolicies =>
                throwException, por lo que sólo se captura en ese caso*/

                if(transPolicy = "throwException")

                {

                        fsm.println("\n\t\t    /* Al heredar "+exceptionName+"
                        de "+fatherException+" primero se comprueba")

                        fsm.println("\t\t    si la excepción lanzada es del
                        tipo "+exceptionName+" */")

                        fsm.println("\t\t    catch ("+exceptionName+ "
                        exc)\n\t\t    {//se capturan excepciones de
                        tipo "+exceptionName)

                        fsm.println('\t\t
                        System.out.println(exc.getMessage());')

                        fsm.println("\t\t    }")

                }

                fsm.println("\t\t    /* Se comprueba si la excepción lanzada es
                del tipo "+fatherException+" */")

                fsm.println("\t\t    catch (NullPointerException exc)\n\t\t
                {//se capturan excepciones de tipo
                        NullPointerException")

                fsm.println('\t\t    System.out.println("No se puede realizar la
                        transición "+ transition.'+tn.name+' +", al no estar
                        activa la region a la que pertenece");')

                fsm.println('\t\t    System.out.println("Lanzada excepción
                        NullPointerException");')

                fsm.println("\t\t    }")

                fsm.println("\t\t    /* Con el siguiente catch, se asegura que
                no se escape ninguna excepción */")

                fsm.println("\t\t     catch (Exception exc)\n\t\t     {//se capturan el
                        resto de Excepciones")

                fsm.println('\t\t    System.out.println("No se puede realizar la
                transición "+ transition.'+tn.name+');')

                fsm.println('\t\t    System.out.println("Lanzada excepción con
                mensaje:"+ exc.getMessage());')

                fsm.println("\t\t    }")

                println ("\t\t }")

}

fsm.println ("\t // -------------------------------------");

fsm.println("\t public void notifyEnd( ){");
```

```
fsm.println('\t\t System.out.println ("          >>>> Entrada a estado FINAL => FIN DEL SIMULADOR");');

fsm.println('\t\t user.notifyEnd();')

fsm.println("\t } \n"); //Fin notifyEnd()

fsm.println("}//FIN CLASE FSM"); //FIN CLASE FSM

//---------------------------------------------

//---------------CLASE STATE-------------------

//---------------------------------------------

file claseStado(path+"\\"+self.name +"_State.java");

claseStado.println("package "+package+";");

claseStado.println("abstract class "+self.name+ "_State{");


//VARIABLES

claseStado.print("\t enum estado { ")

states -> forEach ( s : MetaModel.State ) {

                    if (position()!=0) claseStado.print (", ")

                    claseStado.print ( s.name )

          }

claseStado.println("};")

claseStado.println("\t estado name;");

claseStado.println("\t "+self.name+"_FSM theFSM;\n")

//FUNCIONES

claseStado.println("\t abstract estado getEstado ();");

claseStado.println("\t void onEntry(){}");

claseStado.println("\t void ejecutar(){}");

claseStado.println("\t void onExit(){}");

claseStado.println("}");//Fin de la clase State

//---------------------------------------------

//---------------CLASES DE CADA ESTADO---------

//---------------------------------------------

//SE CREA UNA CLASE POR CADA ESTADO

var name : String

states -> forEach ( s : MetaModel.State ) {

name = s.name

file name(path+"\\" + self.name + "_"+ name + "State.java")

name.println("package "+package+";");

name.println("class "+self.name+"_"+name+"State extends "+self.name+"_State{");

//SE CREA EL CONSTRUCTOR

name.println("/*Se crea el constructor*/")

name.println("\t "+self.name+"_"+name+"State ("+self.name+"_FSM fsm){");

name.println("\t\t name = estado."+name+";");

name.println("\t\t theFSM = fsm;")

name.println("\t }\n");

//FUNCIONES DE CADA ESTADO

name.println("\t estado getEstado () {\n \t\t return name; \n \t }");

name.println("\t void onEntry(){");

name.println("\t\t // -->>>> write code here...");

name.println('\t\t System.out.println("          >>>> Executing '+name+'.onEntry()...");');

if(s.oclIsTypeOf(MetaModel.FinalState))

          name.println('\t\t theFSM.notifyEnd();');

name.println("\t }");//Fin de la funcion onEntry


/*Las funciones ejecutar y onExit() sólo las ejecutarán los estados que no sean de tipo final*/

if(!s.oclIsTypeOf(MetaModel.FinalState))

{

          name.println("\t void ejecutar(){");

          name.println("\t\t // -->>>> write code here...");

          name.println('\t\t System.out.println("          >>>> Ejecutando '+name+'.ejecutar()");');

          name.println('\t\t System.out.println("          >>>> Estado actual de '+s.owner.name+' : '+s.name+'");');

          name.println("\t }");//Fin de la funcion do

          name.println("\t void onExit(){");

          name.println("\t\t // -->>>> write code here...");
```

```
        name.println('\t\t System.out.println("          >>>> Ejecutando
        '+name+'.onExit()");');

        name.println("\t }");//Fin de la funcion onExit

}

name.println("\t String getString(){");

name.println('\t\t String info = null;');

name.println('\t\t info += name;');

name.println('\t\t return info;');

name.println("\t }");

name.println("}");// FIN DE LA CLASE

}// FIN states -> forEach ( s : MetaModel.State ) {

// --------------------------------------------

// ---------- simulator (main) ----------------

// --------------------------------------------

var opciones : integer;

file simu_adb (path+"\\"+self.name + "_Simulator.java")

simu_adb.println("package "+package+";");

simu_adb.println("public class " + self.name + "_Simulator implements
"+self.name+"_finFSMinterface{");

simu_adb.println("\t int opc;");

simu_adb.println("\t static boolean continuar;");

simu_adb.println("\t "+self.name +"_FSM FSM;")

simu_adb.println("\n\t /*SE CREA EL CONSTRUCTOR*/");

simu_adb.println("\t public "+self.name+"_Simulator(){");

simu_adb.println("\t\t FSM = new " + self.name+"_FSM(this);");

simu_adb.println("\t\t continuar = true;");

simu_adb.println("\t}");

simu_adb.println("\n\t/*Funcion que notificará el fin de la máquina de
estados*/")

simu_adb.println("\t public void notifyEnd(){");

simu_adb.println("\t\t continuar = false; //Para finalizar el bucle
infinito, se iguala su condición a false");

simu_adb.println("\t }");

simu_adb.println("\t public void fire_transition ( int opc ) {");

simu_adb.println("\t switch(opc){");

transitions -> forEach ( t : MetaModel.Transition ) {

        opciones = (count()+1)

        simu_adb.println("\t\t\t case " + opciones + " : FSM."+t.name+
        "();");

        simu_adb.println("\t\t\t\t\t break;");

}

opciones = opciones +1;

simu_adb.println("\t\t\t case "+opciones+" :
FSM.getEstadosActualesToString();")

simu_adb.println("\t\t\t\t\t break;");

simu_adb.println("\t\t\t default : break;");

simu_adb.println("\t\t } //Fin switch");

simu_adb.println("\t } // Fin fire_transition\n");

simu_adb.println("public static void main(String args[]){");

simu_adb.println("\t"+self.name+"_Simulator sm = new
"+self.name+"_Simulator();");

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

simu_adb.println('\t System.out.println (" Bienvenido a ' + self.name +
'_Simulator" );');

simu_adb.println("\t while(continuar){")

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

transitions -> forEach ( t : MetaModel.Transition ) {

        opciones = (count()+1)

        simu_adb.println('\t\t System.out.println ("   ' + opciones + ".-
        " + t.name + '" );');

}

opciones = opciones +1;

simu_adb.println('\t\t System.out.println ("   ' + opciones + '.- Mostrar
estados actuales en pantalla");');

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

simu_adb.println('\t\t System.out.println ("   0.- Finalizar Simulador"
);');
```

```
        simu_adb.println('\t System.out.println ("'+ '-----------------------------
        -------------" );');

        simu_adb.print('\t\t System.out.print ("   Selecciona transición a disparar
        (0-' + opciones +'):  ");');

        simu_adb.println("\n\t\t int elec = Teclado.readInt(  );");

        simu_adb.println('\t System.out.println ("'+ '-----------------------------
        -------------" );');

        simu_adb.println('\t\t if (elec == 0){ \n\t\t\t System.out.println("Ha
        elegido finalizar el simulador.");');

        simu_adb.println("\t\t\t continuar = false; //Si opc=0, salir del bucle
        infinito \n \t\t\t } ");

        simu_adb.println("\t\t sm.fire_transition ( elec );");

        simu_adb.println("\t } // Fin while")

        simu_adb.println(" } // Fin main \n");

        simu_adb.println("} //Fin class");

} // module::main

//----------------------------------------------------------------------------------
module::getTransitions ( ) : List {

/*Este metodo se encargará de recoger las transiciones que no tengan como fuente o destino
un pseudoEstado, a no ser que el pseudoEstado fuente sea de tipo join o el pseudoEstado
destino sea del tipo fork */

        var transitions : List

        var repe : boolean

        var pseudoEstadoValido : boolean

        self.objectsOfType ( MetaModel.Transition )->forEach ( s :
        MetaModel.Transition ) {

                pseudoEstadoValido = false

                if((s.source.oclIsTypeOf(MetaModel.Pseudostate)) and
                (s.source.kind = "JoinState"))

                        pseudoEstadoValido = true

                if((s._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
                and (s._getFeature("target").kind = "ForkState"))

                        pseudoEstadoValido = true


        if((!s.source.oclIsTypeOf(MetaModel.Pseudostate)) and
                (!s._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate)))
                pseudoEstadoValido = true

                if(pseudoEstadoValido)

                {

                        repe = false

                        transitions->forEach(c : MetaModel.Transition)

                        {

                                if(c == s) repe = true

                        }

                        if(!repe){

                                transitions.add ( s )

                        }

                }

        }

        result = transitions

}  // getTransitionNames

//----------------------------------------------------------------------------------

module::getTransitionNames ( ) : List {

        var transitionNames : List

        var transitions : List = getTransitions()

        transitions->forEach(t:MetaModel.Transition) transitionNames.add(t.name)

                result = transitionNames

}  // getTransitionNames

//----------------------------------------------------------------------------------
module:: getStates ( ) : List {

        var states: List

        var repe : boolean

        self.objectsOfType ( MetaModel.State )->forEach ( s : MetaModel.State ) {

                repe = false

                states->forEach(c:MetaModel.State)

                {

                        if(c == s) repe = true

                 }

                 if(!repe){

                        states.add ( s )

                        if(!s.region.isEmpty()){
```

```
                              s.objectsOfType(MetaModel.State)->forEach(eI :
                              MetaModel.State)
                                        {
                                                  states.add(eI)
                                        }
                              }
                    }
          }
          result = states
}  // getStates
//-------------------------------------------------------------------------------------
module:: writeProcedure (transicion : MetaModel.Transition)
 {
          var t3:String = "\t\t\t "
          var t4:String = "\t\t\t\t "
          var sName:String  // source state name
          var tName:String  // target state name
          var trans:String // transition name
          var joinTrans : List
          var regiones : List
          var transitionList:List= getTransitions()->select(t : MetaModel.Transition
          |( t.name = transicion.name ))
          if ( not ( transitionList.isempty ( ) ) )
          {
                    if(!transicion.source.oclIsTypeOf(MetaModel.Pseudostate))
                    /*si la transicion proviene de un pseudoestado
                    no se utiliza un segundo switch*/
                    {
                              println (t3+"switch ("+
                              transicion.owner.name+"Actual.getEstado()){")               }

                    transitionList->forEach(t: MetaModel.Transition)
                    {
                    sName = t.source.name
```

```
                    tName = t._getFeature("target").name
                              trans = t.name
//-------------------------------------------------------------------------------------
        /*Se comprueba si el estado que lanza la transicion es un pseudoEstado,
          será del tipo JOIN, ya que el pseudoEstado fork sólo recibe las transiciones
          lanzadas desde el simulador, pero no las lanza*/
          if((t.source.oclIsTypeOf(MetaModel.Pseudostate)) and
          (t.source.kind = "JoinState"))
                    {
           /*Si la transicion tiene como fuente un pseudoEstado  join,antes de lanzar   la
          transicion,  se  debe  comprobar  si  se  encuentra  en  los  estados  necesarios
          para lanzar la transicion desde JOIN*/
          /*Para  encontrar  estos  estados,  primero  se  buscan  las  transiciones  que
          tienen  como  destino,  el  pseudoEstado  JOIN  desde  el  que  se  ha  lanzado  la
          transicion*/
                    joinTrans = self.objectsOfType(MetaModel.Transition)
                    ->select(jt:MetaModel.Transition |
          (jt._getFeature("target") == t.source))
                    print (t4+"if( ")
                    joinTrans->forEach(jt:MetaModel.Transition)
                    {
                    if(position()!=0)print(" && ")
                              print("("+jt.owner.name+"Actual ==
                    "+jt.source.name+")")
                              }
                              println (" ){")
                              joinTrans->forEach(jt:MetaModel.Transition)
                              {
                              if(jt.source.owner!=jt._getFeature("target").owner)
                                {
          /*Antes de comprobar el camino a seguir por cada transición, se debe salir    del
          estado en el que se encuentra, la actualización del pseudoEstado historico, si lo
          tiene, se realizará al llamar el méto  comprobarLocalización()*/
                              println (t4 +"\t"+ jt.owner.name + "Actual.onExit();");
                              comprobarLocalizacion(jt)
                                        }
          }
          /*Una  vez  se  ha  llegado  al  estado  que  contiene  las  transiciones  que  tienen
          como  objetivo  el  pseudoEstado  Join,  se  sale  de  este  estado  y  se  lanza  la
          transición*/
```

```
                        println (t4 +"\t"+ t._getFeature("target").owner.name +
                        "Actual.onExit();");

                        lanzarTransicion(t)

                        println (t4+" }")

                }//fin if => la fuentes es un pseudoEstado de tipo join

//---------------------------------------------------------------------------------
        else      //la fuente es un estado

                {

/*Se comprueba si el estado que lanza la transicion, contiene una región si es así, antes de
lanzar la transición, debe salir de los estados interiores*/

                        println (t4+"case "+ sName + " : //Se encuentra en
                        el estado "+sName)

                        if(!t.source.region.isEmpty())

                         {

                        regiones =
                        self.objectsOfType(MetaModel.Region)-
                >select(reg:MetaModel.Region | (reg.owner ==
        t.source))

                        regiones->forEach(rI:MetaModel.Region)

                        {

                                getEstadosInteriores(rI)-
                                >forEach(estInt:MetaModel.State)

                                {
                if(!estInt.region.isEmpty())regiones.add(estInt.region)

                                }

                        }

                        while(!regiones.isEmpty())

                        {

                        println (t4 +"\t"+ regiones.last().name +
                        "Actual.onExit();");

/*Se comprueba si la region tiene historico, actualizándo su valor si existe*/

                        if(comprobarHistory(regiones.last()))

                        {

                                println (t4 +"\t"+ regiones.last().name +
                        "History = "+regiones.last().name+"Actual;");

                        }

                        println (t4 +"\t"+ regiones.last().name + "Actual =
                        null;");

                                regiones.remove(regiones.last())

                                }

                              }

/*Ya esta a un mismo nivel de region, por lo que se sale del estado que lanza la transicion
y se lanza la transicion*/

                              println (t4 +"\t"+t.source.owner.name +
                              "Actual.onExit();");

                              lanzarTransicion(t)

                              println (t4+"\tbreak; \n");

                }//else => la fuente es un estado

//---------------------------------------------------------------------------------

/*Ya se han comprobado las dos posibles fuentes que pueden lanzar transiciones. Las
siguientes líneas de código se encargarán de completar el código JAVA, añadiendo la clausula
default si la fuente no es un pseudoEstado, "else" si es un pseudoEstado y añadiendo la
forma de actuar si el objetivo es un estado de tipo final*/

//---------------------------------------------------------------------------------

                if(!t.source.oclIsTypeOf(MetaModel.Pseudostate))

                {

                        println(t4+'default:')

                        println(t4 +'\t System.out.println ("No se puede realizar la
                        transicion '+trans+', se continua en estado: ");')

                        println(t4 + '\t getEstadosActualesToString();')

/*La siguiente excepcion sólo se lanzará si a la hora de crear el modelo, se elige la opción
UnexpectedTransPolicies => throwException*/

                        if(transPolicy = "throwException")

                        {

                                println(t4 +'\t throw new '+exceptionName+"();")

                        }

/*si la transicion proviene de un pseudoestado no se utiliza un segundo switch, por lo que
la siguiente llave no es usada*/

                println (t3+"}");

                                }

//---------------------------------------------------------------------------------

/*Si la transicion viene de un pseudo estado, al sustituir el segundo "switch", por

el condicional "if", en lugar  de la sentencia "default", se utiliza "else"*/

               else
```

```
        {
                println(t4+'else { ')
                println(t4 +'\t System.out.println ("No se puede realizar la
                transicion '+trans+', se continua en estado: ");');
                println(t4 + '\t getEstadosActualesToString();')
/*La siguiente excepcion sólo se lanzará si a la hora de crear el modelo, se elige la opción
UnexpectedTransPolicies => throwException*/
                if(transPolicy = "throwException")
                {
                        println(t4 +'\t throw new '+exceptionName+"();")
                }
                println(t4+'}')
          }
//-------------------------------------------------------------------------------
                }//fin forEach(por cada transicion)
        }//fin if la lista no esta vacia
}  // writeProcedure
//-------------------------------------------------------------------------------
module:: getEstadosInteriores (regionActual : MetaModel.Region):List  {
        var estadosInteriores : List
        estadosInteriores = self.objectsOfType(MetaModel.State)-
>select(s:MetaModel.State | (s.owner == regionActual))
        result = estadosInteriores
}  // getEstadosInteriores
//-------------------------------------------------------------------------------
module:: lanzarTransicion(trans2Fire : MetaModel.Transition): void
{
        /*Declaracion de variables*/
        var tName: String = trans2Fire._getFeature("target").name
        var trans: String = trans2Fire.name
        var t4:String = "\t\t\t\t ";
        var forkTrans : List
        var nextState : MetaModel.State
        /*Condicionantes de lanzamiento*/
        /*Se comprueba si la transicion es externa*/
        if(trans2Fire.kind = "external")
        {
          /*Se comprueba si el destino es un pseudoEstado*/
           if(!trans2Fire._getFeature("target").oclIsTypeOf(MetaModel.Pseudo
                state))
                {/*El destino es un estado*/
                        println (t4 +"\tfire."+trans+"_fire();");
                        comprobarLocalizacion(trans2Fire)
                 }//fin el destino es un estado
//-------------------------------------------------------------
                else //el destino es un pseudoEstado
                {
/*En el caso de que el destino sea un pseudoEstado, este pseudoEstado ha de ser de tipo
fork, ya que a los pseudoEstados iniciales no llegan transiciones y al pseudoEstado join, se
llega automaticamente, una vez se llega a los estados necesarios*/
        /*CASO FORK*/
           if((trans2Fire._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
        and (trans2Fire._getFeature("target").kind = "ForkState"))
             {
 /*Si es de tipo fork, se recogen las transiciones que tienen como fuente este pseudoEstado
y se pasa a los estados que se indiquen*/
/*Al tratarse de un pseudoEstado fork, se ejecutarán varias transiciones, esto conlleva  un
cambio de región, ya que en una misma región no se puede estar en más de un estado */
 /*Antes de comprobar la localizacion, se realiza la entrada al estado al se dirige, ya que
esta entrada sólo debe realizarse una vez y no una vez por cada transición*/
                 println (t4 +"\tfire."+trans+"_fire();");
         forkTrans = self.objectsOfType(MetaModel.Transition)-
        >select(ft:MetaModel.Transition |
        (trans2Fire._getFeature("target")== ft.source))
                 nextState = forkTrans.first()._getFeature("target")
         while(nextState.owner!=trans2Fire.source.owner)
                        nextState=nextState.owner
                 println (t4 +"\t"+ trans2Fire.source.owner.name +"Actual = " +
                nextState.name + ";");
                println (t4 +"\t"+ trans2Fire.source.owner.name +
                        "Actual.onEntry();");
               println (t4 +"\t"+ trans2Fire.source.owner.name +
                        "Actual.ejecutar();");
```

```
                    forkTrans->forEach(ft:MetaModel.Transition)

                    {

                            println (t4 +"//----------------------------");

                            comprobarLocalizacion(ft)

                    }//fin forEach(cada transicion Fork)

                   }//fin caso fork

             }//fin else, el destino es un pseudoEstado

         } //fin es una transicion externa

//-------------------------------------------------------------------------------

         /*transición interna*/

         else

         {

                 println (t4 +"\tfire."+trans+"_fire();");

                 println(t4 +"\t"+'System.out.println("           >>>> Currently
                 the '+trans2Fire.owner.name+'Region state is:
         '+tName+'");')

         }

}

//----------------------------------------------------------------------------

module:: getPrimerEstadoName (region : MetaModel.Region) : String

{

         var primerEstadoName : String

         self.objectsOfType ( MetaModel.Transition )->forEach ( s :
         MetaModel.Transition )

         {

                 if((s.source.oclIsTypeOf(MetaModel.Pseudostate))and
                 ((s.source.kind = "InitialState") or (s.source.kind =
         "HistoryState")) and (s.source.owner == region))

                 {

                         primerEstadoName = s._getFeature("target").name

                 }

         }

         result = primerEstadoName

}
```

```
//-----------------------------------------------------------------------------------
module:: comprobarLocalizacion(trans2Check : MetaModel.Transition): void

{

/*Este metodo lo utiliza cada vez que se lance una transición y se encargará de comprobar si
la transicion,conduce a una region más interna o más externa y actuar en consecuencia*/

/*Recibe como parametros, la transicion deseada*/

         var t4 = "\t\t\t\t"

         var fuenteList : List

         var destinoList : List

         var fuente : MetaModel.State

         var destino : MetaModel.State

         var regiones :  List

         var interior : boolean

/*Se rellenan las listas con todos los estados "propietarios" de la fuente y el destino*/

         fuente = trans2Check.source

         destino = trans2Check._getFeature("target")

/*Se realiza la siguiente operación para introducir también las transiciones entre estados
que esten en topRegion. Ya que en este caso, no entraría en el bucle while. */

/*Se añaden los estados de los que se debe entrar y salir para realizar la transicion*/

          while((fuente.name != regionSM)or(destino.name!=regionSM))

          {

                  if(fuenteList.last().name != regionSM)

                          {

                                 fuenteList.add(fuente)

                          if(fuente.owner.name == regionSM)

                                  fuenteList.add(fuente.owner)

                          fuente = fuente.owner

              }

              if(destinoList.last().name != regionSM)

                   {

                          destinoList.add(destino)

                          if(destino.owner.name == regionSM)

                                  destinoList.add(destino.owner)
```

```
                              destino = destino.owner
                    }
            }
/*Se han rellenado ambas listas hasta llegar a topRegion, pero habrá transiciones en las que no se tiene que llegar a topRegion, por lo que se comparan ambas listas, y borrando las entradas que coincidan, ya que  serán estados comunes, de los que no tendrá que entrar ni salir*/
          interior = true
          while(interior)
          {
                    if(destinoList.last() == fuenteList.last())
                    {
                      destinoList.remove(destinoList.last())
                      fuenteList.remove(fuenteList.last())
                    }
                    else interior = false /*Cuando dejen de ser iguales, se sale del
                              bucle*/
          /*Si despues de borrar, alguna de las listas está vacía, se sale del
          bucle*/
                    if(fuenteList.isEmpty() or destinoList.isEmpty())interior = false
          }
/*Una vez se han rellenado ambas listas, lo primero que se debe hacer antes de entrar en los estados "propietarios" del objetivo, es salir de los estados "propietarios" de la fuente*/

/*El primer estado del que se debe salir es el estado fuente, que estará almacenado en las primeras posiciones de la lista*/
/*Si la transición se dirige a un pseudoEstado de tipo join, se tendrá un estado común, del que tendrán que salir todas las transiciones del pseudoEstado.Este estado será el que se encuentre en la misma región que el destino y corresponde con el último introducido en la lista, así que se borra y se sale de este estado cuando se vuelva a la invocación de este método, comprobarLocalizacion()*/

          if((trans2Check._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
          and (trans2Check._getFeature("target").kind == "JoinState"))
                    fuenteList.remove(fuenteList.last())
          while(!fuenteList.isEmpty())
          {
                    if(fuenteList.first().oclIsTypeOf(MetaModel.State))
                    {
/*Se comprueba si el estado del que se va a salir, en el recorrido de la transicion, contiene una región si es así, antes de salir del estado, debe salir de los estados interiores, a no ser que sea el estado que lanza la transición, ya que en este caso ya se ha salido de sus estados interiores, antes de lanzar la transición*/
                              if((fuenteList.first() !=
                              trans2Check.source)and
                    (!fuenteList.first().region.isEmpty()))
                                {
                                        regiones =
                              self.objectsOfType(MetaModel.Region)->select
                    (reg:MetaModel.Region | (reg.owner ==
                    fuenteList.first()))
                                        regiones->forEach(rI:MetaModel.Region)
                                        {
                                                  getEstadosInteriores(rI)-
                                        >forEach(estInt:MetaModel.State)
                                                            {

                    if(!estInt.region.isEmpty())regiones.add(estInt.region)
                                                            }
/*En el caso de que el estado sea el estado fuente, ya ha echo esta comprobacion y se ha salido del estado antes de lanzar la transicion, por lo que se asegura que no vuelva a salir de este estado*/
                    regiones.remove(trans2Check.source.owner)
                                        }
                              while(!regiones.isEmpty())
                              {
                                        println (t4 +"\t"+  regiones.last().name +
                              "Actual.onExit();");
/*Se comprueba si la region tiene historico, actualizándo su valor si existe*/
                              if(comprobarHistory(regiones.last()))
                              {
                                        println (t4 +"\t"+  regiones.last().name +
                              "History = "+regiones.last().name+"Actual;");
                              }
                              println (t4 +"\t"+ regiones.last().name +
                    "Actual = null;");
                              regiones.remove(regiones.last())
                              }
```

```
                    }

/* Si la fuente que lanza la transición es de tipo State, antes de lanzar la transición ya se ejecuta su función .onExit() antes de lanzar la transición, por lo que no se vuelve a lanzarla. Sí se establece su estado actual a null por tratarse de un cambio de región debido a una salida*/

                    if(fuenteList.first().owner != trans2Check.source.owner)

                              println (t4 +"\t"+ fuenteList.first().owner.name +
                    "Actual.onExit();");

/*Sólo se deben poner a null los estados de las regiones de las que se sale*/

                    if(fuenteList.first().owner !=
                    trans2Check._getFeature("target").owner)

                    {

/*Si la transición provoca una salida de la región,se comprueba si la region tiene historico, actualizándo su valor si existe*/

          if(comprobarHistory(fuenteList.first().owner))

                              println (t4 +"\t"+ fuenteList.first().owner.name +
                              "History = "+fuenteList.first().owner.name+
                    "Actual;");

                              interior = false

                              destinoList->forEach(dest : MetaModel.State)

                              {

                                        if(dest.owner ==
                              fuenteList.first().owner)interior = true

                              }

/*Si la transicion se realiza a un estado interior de algun estado de la region en la que se encuentra, el valor actual de la region será este estado, por lo que no se actualiza a null*/

                              if(!interior)

                                        println (t4 +"\t"+
                              fuenteList.first().owner.name + "Actual =
                    null;");

                        }

/*Al salir de una region interior, se establece su valor de estado actual a null,

para que no permita transiciones erroneas. Tener en cuenta, que cuando se encuentra en la misma región que el estado destino, no se sale de ella, se cambia su estado actual*/

                    }

                              fuenteList.remove(fuenteList.first())

             }//fin while
```

```
/*Una vez se ha salido de los estados necesarios, se va entrando a traves de los estados que contienen el objetivo*/

/*Si la transición proviene un pseudoEstado de tipo fork, se tiene un estado común, al que tendrán que entrar todas las transiciones del pseudoEstado.Este estado será el que se encuentre en la misma región que la fuente y corresponde con el ultimoo elemento de la lista,así que se borra y se entra a este estado antes de la invocación de este método, comprobarLocalizacion()*/

          if((trans2Check.source.oclIsTypeOf(MetaModel.Pseudostate))and(trans2Check.s
          ource.kind == "ForkState"))

                    destinoList.remove(destinoList.last())


            while(!destinoList.isEmpty())

            {
          if((destinoList.last().oclIsTypeOf(MetaModel.State))or(destinoList.last().o
          clIsTypeOf(MetaModel.FinalState)))

          {

          if(destinoList.last().owner!=trans2Check._getFeature("target").owner)

              {

                    if(comprobarHistory(destinoList.last().owner))

                              println (t4 +"\t"+ destinoList.last().owner.name +
                    "Actual = "+destinoList.last().owner.name+ "History;");

                    else

                              println (t4 +"\t"+ destinoList.last().owner.name +
                    "Actual =
          "+getPrimerEstadoName(destinoList.last().owner)+
          "Actual;");

              }

              else

                    println (t4 +"\t"+ destinoList.last().owner.name + "Actual =
          "+destinoList.last().name+ ";");


              println (t4 +"\t"+ destinoList.last().owner.name +
          "Actual.onEntry();");

              println (t4 +"\t"+ destinoList.last().owner.name +
          "Actual.ejecutar();");

          /*se comprueba si el estado al que se pasa, contiene más regiones*/

              if(!destinoList.last().region.isEmpty())

                  {

/*Se recogen todas las regiones, cuyo propietario es el estado al que se ha entrado*/
```

```
                    regiones = self.objectsOfType(MetaModel.Region)-
                    >select(r:MetaModel.Region | (r.owner ==
          destinoList.last()))

                    regiones->forEach(regionActual:MetaModel.Region)

                     {

/*Como se ha entrado en una región, también se entra directamente al estado al que apunte su
pseudoEstado,por lo que se debe ejecutar sus funciones _onEntry() y _do()*/

                    if(comprobarHistory(regionActual))

                    {

/*Si la region tiene un pseudoEstado historico, al entrar a la region, se accede

al estado al que apunta el historico*/

                              println (t4 +"\t"+regionActual.name+"Actual =
                    "+regionActual.name+"History;");

                      }

                      else

                      {

/*Si la region region no tiene historico, se accede al estado apuntado por el pseudoEstado
inicial */

                              println (t4 +"\t"+regionActual.name+"Actual =
                              "+getPrimerEstadoName(regionActual)+ ";");

                      }

                      println (t4 +"\t"+ regionActual.name + "Actual.onEntry();");

                      println (t4 +"\t"+ regionActual.name + "Actual.ejecutar();");

                      }//fin forEach regiones

                     }//fin el estado tiene regiones interiores

                    }

          destinoList.remove(destinoList.last())

          }//fin while(dl:MetaModel.State)

}

//--------------------------------------------------------------------------------------

module:: comprobarHistory (region2Check : MetaModel.Region):boolean  {

          var historico : boolean = false

          var pseudoEstados : List = self.objectsOfType(MetaModel.Pseudostate)

          pseudoEstados->forEach(hist:MetaModel.Pseudostate)

          {

                    if((hist.kind == "HistoryState")and(hist.owner==region2Check))

                              historico = true

          }

          result = historico

}  // comprobarHistory

//--------------------------------------------------------------------------------------

} // texttransformation MOFScriptExample
```

B. 2 Código MOFSCRIPT. Ts2FOO.

```
/**
 * transformation SM_ModeltoJava_Ts2FOO
 */
texttransformation Ts2FOO (in MetaModel:StateMLplus) {
/*Se declaran estas variables globales, porque se usarán tanto en el main, como en los métodos utilizados fuera de main*/
        var exceptionName : String = "transitionErrorException" //Nombre de la excepcion creada para transiciones erroneas
        var fatherException :String = "NullPointerException" //Nombre de la excepcion de JAVA de la cual hereda
        var regionSM : String = self.topRegion.name
        var transPolicy : String = self.unexpectedTransPolicy
  MetaModel.StateMachine::main(){
        var path : String = "StateMLplus/"+self.name+"/Ts2FOO"
        var package : String = "StateMLplus."+self.name+".Ts2FOO"
        var states : List = getStates ()
        var transitions : List = getTransitions()
        var transitionNames:List = getTransitionNames ( )
        var regiones : List = self.objectsOfType(MetaModel.Region)

        var historyPseudo : List = self.objectsOfType(MetaModel.Pseudostate)->select(ph:MetaModel.Pseudostate | (ph.kind =="HistoryState"))
        //--------------------------------------------
        //-------CLASE transitionErrorException--------
        //--------------------------------------------
        /*Esta clase sólo se creará si a la hora de crear el modelo, se elige la opción UnexpectedTransPolicies => throwException*/
        if(transPolicy = "throwException")
        {
                file excep(path+"\\"+exceptionName+".java");
                excep.println("package "+package+";");
                excep.println("public class "+exceptionName+ " extends "+fatherException+"{");
                excep.println("\t public "+exceptionName+"()\n \t {");
                excep.println('\t\t super("Lanzada excepción '+exceptionName+'");');
                excep.println("\t }");
                excep.println("}");
        }
        //--------------------------------------------
        //-----------interfaz finFSM------------------
        //--------------------------------------------
        /*Esta interfaz será implementada por FSM, e incluirá el método mediante el cuál un estado final notificará que se ha accedido a este tipo de estado.Esta proceso se llama proceso de CALLBACK*/
        file finInterf(path+"\\"+self.name+"_finFSMinterface.java");
        finInterf.println("package "+package+";");
        finInterf.println("public interface "+self.name+ "_finFSMinterface{");
        finInterf.println("\t public void notifyEnd();");
        finInterf.println("}");
        //--------------------------------------------
        //------INTERFAZ transitioInterface------------
        //--------------------------------------------
        file trs(path+"\\"+self.name+"_transitionInterface.java")
        trs.println("package "+package+";");
        trs.println("public interface "+self.name +"_transitionInterface{");
        trs.print("\t public enum transition { ")
                transitionNames -> forEach ( t : String ) {
                        if (position()!=0) trs.print (", ")
                        trs.print ( t )
                }
        trs.println("};\n")
        trs.println("\t/*el siguiente método lanzará una excepción NullPointerException, si la transición")
        trs.println("\tque se desea lanzar pertenece a una región que todavía no ha sido activada*/")
        trs.print("\t public void fireTransition(transition t) throws NullPointerException")
```

```
/*La siguiente excepcion sólo se lanzará si a la hora de crear el modelo,      se
elige la opción UnexpectedTransPolicies => throwException*/

if(transPolicy = "throwException")

          {

          trs.print(", "+exceptionName)

          }

trs.println(";  //Esta funcion será invocada por el simulador\n")

trs.println("}"); //FIN INTERFACE TRANSITION

//---------------------------------------------

//---------------CLASE FIRE--------------------

//---------------------------------------------

file fire(path+"\\"+self.name+"_fireTransition.java")

fire.println("package "+package+";");

fire.println("class "+self.name +"_fireTransition{");

fire.println("\n\t /*ESTA CLASE CONTIENE LAS IMPLEMENTACIONES DE LAS
FUNCIONES");

fire.println("\t DISPAROS DE LAS TRANSICIONES*/");

fire.println("\n\t /*EL USUARIO INSERTARÁ EN ESTAS FUNCIONES,");

fire.println("\t EL CÓDIGO DESEADO PARA CADA DISPARO*/");

/*Funciones que simulan el disparo de las transiciones*/

transitions->forEach(trans : MetaModel.Transition)

{

          fire.println("\t void "+trans.name+"_fire(){");

          fire.println ('\t\t System.out.println (" \t\t>>>> Ejecutando
          '+trans.name+'_fire...");')

          fire.println("\t}");

}

fire.println("}");

//---------------------------------------------

//---------------CLASE FSM---------------------

//---------------------------------------------

file fsm(path+"\\"+self.name+"_FSM.java")

fsm.println("package "+package+";");

fsm.println("public class "+self.name+"_FSM implements
"+self.name+"_transitionInterface {");

/*Se utiliza un objeto de la clase fireTransition, para simular el   disparo     de
las transiciones*/

fsm.println("\n\t/*Se utiliza un objeto de la clase fire, para simular el
disparo de las transiciones*/");

fsm.println("\t "+self.name+"_fireTransition fire = new
"+self.name+"_fireTransition();\n");


/*SE CREA UN OBJETO POR CADA POSIBLE ESTADO*/

fsm.println("\n\t /*Se crea un objeto por cada estado*/");

states->forEach(s:MetaModel.State)

{

          fsm.println("\t final "+self.name+"_"+s.name+"State "+s.name+";")

}

/*Ahora se evaluan los pseudoEstados, serán de tipo join*/

states -> forEach (p : MetaModel.Pseudostate)

          fsm.println("\t final "+self.name+"_"+p.name+"State "+p.name+";")

 /*Se crean las variables que contendrán el estado actual de cada región*/

regiones->forEach(reg:MetaModel.Region)

{

          fsm.println("\t protected "+self.name+"_State
          "+reg.name+"Actual;");

}

/*Variables para las regiones que tengan historico*/

historyPseudo->forEach(ph:MetaModel.Pseudostate)

{

          fsm.print("\t protected "+self.name+"_State
          "+ph.owner.name+"History;")

}

fsm.println("\n\t /*Se crea objeto de tipo finFSMinterface*/");

fsm.println("\t "+self.name+"_finFSMinterface user;")

fsm.println("\n\t /*SE CREA EL CONSTRUCTOR*/")

fsm.println("\n\t public "+self.name+"_FSM("+self.name+"_finFSMinterface
userNot){")
```

```
fsm.println("\n\t /*SE INICIALIZAN LOS ESTADOS*/")

states->forEach(s:MetaModel.State)

fsm.println("\t\t "+s.name+" = new "+self.name+"_"+s.name+"State(this);")

states->forEach(p:MetaModel.Pseudostate)

fsm.println("\t\t "+p.name+" = new "+self.name+"_"+p.name+"State(this);")

fsm.println("\n\t /*Se inicializa topRegion y las regiones que tengan
historicos*/")

fsm.println("\t\t "+regiones.first().name+"Actual =
"+getPrimerEstadoName(regiones.first())+";")

/*Se crea una variable para cada region que tenga un pseudoEstado historico */

historyPseudo->forEach(ph:MetaModel.Pseudostate)

{

          fsm.print("\t\t "+ph.owner.name+"History =
          "+getPrimerEstadoName(ph.owner)+";")

}

fsm.println("\n\t/*Se inicializa el objeto que notificará el fin de la
maquina de estados*/")

fsm.println("\t\t user = userNot;");

fsm.println("\n\t }")//fin del constructor

fsm.println("\n\t/*Se crea un tipo de datos enumerado que contendrá los
estados actuales de cada región*/");

fsm.println("\t public enum region {");

regiones->forEach(reg:MetaModel.Region)

{

  /*Solo se inicializa topRegión, ya que será a la que se entre nada más
poner en funcionamiento la máquina de estados. Las demás variables se
inicializan a null*/

          if(position() ==0)

                    fsm.print("\t\t
                    "+reg.name+"("+self.name+
                    "_State.estado."+getPrimerEstadoName(reg)
                    +")")

          if(position() !=0)

          {

                    fsm.print(",\n")

                    fsm.print("\t\t "+reg.name+"(null)")

          }

}

fsm.println(";")

fsm.println("\n\t\t public "+self.name+"_State.estado
currentState;/*Variable que almacenará el estado actual*/")

fsm.println("\n\t /*Se crea un constructor para inicializar la varible de
estado actual, currentState*/")

fsm.println("\t\t region ("+self.name+"_State.estado estadoIni) {");

fsm.println("\t\t\t currentState = estadoIni;");

fsm.println("\t\t }");

fsm.println("\t }");

fsm.println("\n\t/*Se crea una instancia del enumerado region, para poder
acceder a sus campos*/");

fsm.println("\t/*Se debe inicializar a cualquier valor del enumerado, da
igual cual sea*/")

fsm.println("\t region estadoActual = region."+regiones.first().name+";")

/*FUNCION QUE DEVOLVERÁ EL ESTADO ACTUAL DE LA MÁQUINA DE ESTADOS.

DEVOLVERÁ UN TIPO DE DATOS ENUMERADO, QUE CONTENDRÁ EL ESTADO ACTUAL DE
CADA REGIÓN*/

fsm.println("\n\t /*FUNCION QUE DEVOLVERÁ EL ESTADO ACTUAL DE LA MÁQUINA DE
ESTADOS.");

fsm.println("\t DEVOLVERÁ UN TIPO DE DATOS ENUMERADO, QUE CONTENDRÁ EL
ESTADO ACTUAL DE CADA REGIÓN*/");

fsm.println("\t public region getEstadoActual (){");

fsm.println("\t\t return estadoActual;")

fsm.println("\t }");


/*FUNCION QUE SE ENCARGARÁ DE ACTUALIZAR EL VALOR DEL ENUMERADO QUE
CONTIENE LOS ESTADOS ACTUALES*/

fsm.println("\n\t/*FUNCION QUE SE ENCARGARÁ DE ACTUALIZAR EL VALOR DEL
ENUMERADO QUE CONTIENE LOS ESTADOS ACTUALES*/");

fsm.println("\t public void actualizarEstados(){")

regiones->forEach(reg:MetaModel.Region)

{

          fsm.println("\t\t if("+reg.name+"Actual != null)");

          fsm.println("\t\t\t estadoActual."+reg.name+".currentState =
          "+reg.name+"Actual.name;")

          fsm.println("\t\t else ")
```

```
        fsm.println("\t\t\t estadoActual."+reg.name+".currentState =
        null;")

}

fsm.println("\t }")

/*FUNCION QUE DEVOLVERÁ LOS ESTADOS ACTUALES DE TODAS LAS REGIONES EN
FORMATO STRING*/

fsm.println("\n\t /*FUNCION QUE DEVOLVERÁ LOS ESTADOS ACTUALES DE TODAS LAS
REGIONES*/");

fsm.println("\t public void getEstadosActualesToString(){")

        regiones->forEach(reg:MetaModel.Region)

                {

                        fsm.println("\t\t\t\t if("+reg.name+"Actual
                        != null)");

                        fsm.println('\t\t\t\t\t
                        System.out.println(">>>>>>>>>>
                Region '+reg.name+' en estado:"
        +'+reg.name+'Actual.name);');

                        fsm.println("\t\t\t\t else");

                        fsm.println('\t\t\t\t\t
                        System.out.println(">>>>>>>>>>
                Region '+reg.name+' no activa");');

                }

fsm.println("\t }")


fsm.print("\n\t public void fireTransition(transition t)throws
NullPointerException ")

/*La siguiente excepción sólo se lanzará si se elige la opción
unexpectedTransPolicy => throwException*/

if(transPolicy = "throwException")

        {

fsm.print(","+exceptionName)

        }

fsm.println(" {")

fsm.println("\t   try\n \t   {//Parte de código que puede provocar
excepciones")

fsm.println ("\t // ------------------------------------");

fsm.println("\t\t switch (t){")

//FUNCIONES PARA LAS TRANSICIONES

transitions -> forEach ( tn : MetaModel.Transition ) {

                writeProcedure ( tn )

        }

fsm.println("\t\t }//Fin switch (t)")//Fin switch(t)

fsm.println ("\t // ------------------------------------");

fsm.println("\t   }//Fin try")

/*La siguiente excepcion sólo se lanzará si a la hora de crear el modelo, se
elige la opción UnexpectedTransPolicies => throwException, por lo que sólo se
captura en ese caso*/

if(transPolicy = "throwException")

        {

        fsm.println("\n \t /* Al heredar "+exceptionName+" de
        "+fatherException+" primero se comprueba")

        fsm.println("\t si la excepción lanzada es del tipo
        "+exceptionName+" */")

        fsm.println("\tcatch ("+exceptionName+ " exc)\n\t {//se captura
        excepciones de tipo "+exceptionName)

        fsm.println('\t\t System.out.println(exc.getMessage());')

        fsm.println("\t}")

        }


fsm.println("\t /* se comprueba si la excepción lanzada es del tipo
"+fatherException+" */")

fsm.println("\tcatch (NullPointerException exc)\n\t {//se captura
excepciones de tipo NullPointerException")

fsm.println('\t\t System.out.println("No se puede realizar la transición
"+t+", al no estar activa la region a la que pertenece");')

fsm.println('\t\t System.out.println("Lanzada excepción
NullPointerException");')

fsm.println("\t}")

fsm.println("\t /* Con el siguiente catch, se asegura que no se
escape ninguna excepción */")

fsm.println("\tcatch (Exception exc)\n\t {//se capturan el resto de
Excepciones")

fsm.println('\t\t System.out.println("No se puede realizar la transición
"+t);')

fsm.println('\t\t System.out.println("Lanzada excepción con mensaje:"+
exc.getMessage());')
```

```
fsm.println("\t }")

fsm.println ("\t}//Fin fireTransition \n")//fin fireTransition

fsm.println ("\t // -------------------------------------\n");
fsm.println("\t public void notifyEnd( ){");
fsm.println('\t\t System.out.println ("          >>>> Entrada a estado
FINAL => FIN DEL SIMULADOR");');
fsm.println('\t\t user.notifyEnd();')
fsm.println("\t } \n"); //Fin notifyEnd()

fsm.println("}//FIN CLASE FSM"); //FIN CLASE FSM
//-----------------------------------------
//---------------CLASE STATE-------------
//---------------------------------------------
file claseStado(path+"\\"+self.name +"_State.java");
claseStado.println("package "+package+";");
claseStado.println("abstract class "+self.name+ "_State{");
//VARIABLES
claseStado.print("\t enum estado { ")
states -> forEach ( s : MetaModel.State ) {
                    if (position()!=0) claseStado.print (", ")
                    claseStado.print ( s.name )
          }
claseStado.println("};")
claseStado.println("\t estado name;");
claseStado.println("\t "+self.name+"_FSM theFSM;\n")
//FUNCIONES
claseStado.println("\t abstract estado getEstado ();");
claseStado.println("\t void onEntry(){}");
claseStado.println("\t void ejecutar(){}");
claseStado.println("\t void onExit(){}");
claseStado.println("}");//Fin de la clase State
```

```
//---------------------------------------------
//---------------CLASES DE CADA ESTADO---------
//---------------------------------------------
//SE CREA UNA CLASE POR CADA ESTADO
var name : String
states -> forEach ( s : MetaModel.State ) {
name = s.name
file name(path+"\\" + self.name + "_"+ name + "State.java")
name.println("package "+package+";");
name.println("class "+self.name+"_"+name+"State extends
"+self.name+"_State{");
//SE CREA EL CONSTRUCTOR
name.println("/*Se crea el constructor*/")
name.println("\t "+self.name+"_"+name+"State ("+self.name+"_FSM fsm){");
name.println("\t\t name = estado."+name+";");
name.println("\t\t theFSM = fsm;")
name.println("\t }\n");
//FUNCIONES DE CADA ESTADO
name.println("\t estado getEstado () {\n \t\t return name; \n \t
}");
name.println("\t void onEntry(){");
name.println("\t\t // -->>>> write code here...");
name.println('\t\t System.out.println("          >>>> Executing
'+name+'.onEntry()...");');
if(s.oclIsTypeOf(MetaModel.FinalState))
          name.println('\t\t theFSM.notifyEnd();');
name.println("\t }");//Fin de la funcion onEntry
/*Las funciones ejecutar y onExit() sólo las ejecutarán los estados que no
sean de tipo final*/
if(!s.oclIsTypeOf(MetaModel.FinalState))
{
          name.println("\t void ejecutar(){");
          name.println("\t\t // -->>>> write code here...");
```

```
          name.println('\t\t System.out.println("          >>>> Ejecutando
          '+name+'.ejecutar()");');

          name.println('\t\t System.out.println("          >>>> Estado
          actual de '+s.owner.name+' : '+s.name+'");');

          name.println("\t }");//Fin de la funcion do


          name.println("\t void onExit(){");


          name.println("\t\t // -->>>> write code here...");

          name.println('\t\t System.out.println("          >>>> Ejecutando
          '+name+'.onExit()");');

          name.println("\t }");//Fin de la funcion onExit

}

name.println("\t String getString(){");

name.println('\t\t String info = null;');

name.println('\t\t info += name;');

name.println('\t\t return info;');

name.println("\t }");

name.println("}");// FIN DE LA CLASE

}// FIN states -> forEach ( s : MetaModel.State ) {

//----------------------------------------------

//---------------SIMULADOR---------------------

//----------------------------------------------

var opciones : integer

file simu_adb (path+"\\"+self.name+"_Simulator.java")

simu_adb.println("package "+package+";");

simu_adb.println("public class "+self.name+"_Simulator implements
"+self.name+"_finFSMinterface{");

simu_adb.println("\t int opc;");

simu_adb.println("\t static boolean continuar;");

simu_adb.println("\t "+self.name+"_FSM fsmObj;")

/*SE CREA EL CONSTRUCTOR*/

simu_adb.println("\n\t /*SE CREA EL CONSTRUCTOR*/");

simu_adb.println("\t public "+self.name+"_Simulator(){");

simu_adb.println("\t\t continuar = true;");

simu_adb.println("\t\t fsmObj = new "+self.name+"_FSM(this);");
                              //Se utiliza composicion \n");

simu_adb.println("\t }");

simu_adb.println("\n\t /*FUNCION QUE NOTIFICARA */");

simu_adb.println("\t public void notifyEnd(){");

simu_adb.println('\t\t continuar = false;');
simu_adb.println('\t }');

simu_adb.println("\t public void fire_transition ( int opc ){");

simu_adb.println("\t switch(opc){");

transitionNames -> forEach ( t : String ) {

          opciones = (count()+1)

          simu_adb.println("\t\t case " + opciones + " :
          fsmObj.fireTransition("+
self.name+"_FSM.transition."+t+");");

          simu_adb.println("\t\t\t break;");

}

opciones = opciones +1;

simu_adb.println("\t\t case "+opciones+":
fsmObj.getEstadosActualesToString();");

simu_adb.println("\t\t\t break;");

simu_adb.println("\t\t default :")

simu_adb.println('\t\t\t break;');

simu_adb.println("\t\t } //Fin switch");

simu_adb.println("\t } // Fin fire_transition\n");

simu_adb.println("public static void main(String args[]){");

simu_adb.println("\t"+self.name+"_Simulator sm = new
"+self.name+"_Simulator();");

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

simu_adb.println('\t System.out.println (" Bienvenido a ' + self.name +
'_Simulator" );');

simu_adb.println("\t while(continuar){")

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

transitions -> forEach ( t : MetaModel.Transition ) {
```

```
                opciones = (count()+1)

                simu_adb.println('\t\t System.out.println ("   ' + opciones + ".-
                " + t.name + '" );');                                           }

        opciones = opciones +1;

        simu_adb.println('\t\t System.out.println ("   ' + opciones + '.- Mostrar
        estados actuales en pantalla");');

        simu_adb.println('\t System.out.println ("'+ '-----------------------------
        -------------" );');

        simu_adb.println('\t\t System.out.println ("   0.- Finalizar Simulador"
        );');

        simu_adb.println('\t System.out.println ("'+ '-----------------------------
        -------------" );');

        simu_adb.print('\t\t System.out.print ("   Selecciona transición a disparar
        (0-' + opciones +'):  ");');

        simu_adb.println("\n\t\t int elec = Teclado.readInt(  );");

        simu_adb.println('\t System.out.println ("'+ '-----------------------------
        -------------" );');

        simu_adb.println('\t\t if (elec == 0){ \n\t\t\t System.out.println("Ha
        elegido finalizar el simulador.");');

        simu_adb.println("\t\t\t continuar = false; //Si opc=0, salir del bucle
        infinito \n \t\t\t } ");

        simu_adb.println("\t\t sm.fire_transition ( elec );");

        simu_adb.println("\t } // Fin while")

        simu_adb.println(" } // Fin main \n");

        simu_adb.println("} //Fin class");

} // module::main

//-------------------------------------------------------------------------------
module::getTransitions ( ) : List {

/*Este metodo se encargará de recoger las transiciones que no tengan como fuente o destino
un pseudoEstado, a no ser que el pseudoEstado fuente sea de tipo join o el pseudoEstado
destino sea del tipo fork */

        var transitions : List

        var repe : boolean

        var pseudoEstadoValido : boolean

        self.objectsOfType ( MetaModel.Transition )->forEach ( s :
        MetaModel.Transition ) {

        pseudoEstadoValido = false

        if((s.source.oclIsTypeOf(MetaModel.Pseudostate)) and (s.source.kind =
        "JoinState"))

                pseudoEstadoValido = true

        if((s._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate)) and
        (s._getFeature("target").kind = "ForkState"))

                pseudoEstadoValido = true

        if ((!s.source.oclIsTypeOf(MetaModel.Pseudostate))and
        (!s._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate)))
        pseudoEstadoValido = true

        if(pseudoEstadoValido)

        {

                repe = false

                transitions->forEach(c : MetaModel.Transition)

                {

                        if(c == s) repe = true

                }

                if(!repe){

                        transitions.add ( s )

                        }

                }

        }

        result = transitions

}  // getTransitionNames

//-------------------------------------------------------------------------------
module::getTransitionNames ( ) : List {

        var transitionNames : List

        var transitions : List = getTransitions()

        transitions->forEach(t:MetaModel.Transition) transitionNames.add(t.name)

        result = transitionNames

}  // getTransitionNames


//-------------------------------------------------------------------------------
module:: getStates ( ) : List {

        var states: List

        var repe : boolean

        self.objectsOfType ( MetaModel.State )->forEach ( s : MetaModel.State ) {
```

```
                    repe = false
                    states->forEach(c:MetaModel.State)
                    {
                              if(c == s) repe = true
                    }
                    if(!repe){
                              states.add ( s )
                              if(!s.region.isEmpty()){
                                        s.objectsOfType(MetaModel.State)->forEach(eI
                              : MetaModel.State)
                                        {
                                                  states.add(eI)
                                        }
                                }
                    }
          }
          result = states
}  // getStates

//---------------------------------------------------------------------------------------
module:: writeProcedure (transicion : MetaModel.Transition)
 {
          var t3:String = "\t\t\t "
          var t4:String = "\t\t\t\t "
          var sName:String  // source state name
          var tName:String  // target state name
          var trans:String // transition name
          var joinTrans : List
          var regiones : List
          var transitionList:List= getTransitions()->select(t : MetaModel.Transition
          |( t.name = transicion.name ))
          if ( not ( transitionList.isempty ( ) ) )
          {
```

```
                    println ("\t // -------------------------------------")
                    println ("\t\t case " + transicion.name + ":")
                    if(!transicion.source.oclIsTypeOf(MetaModel.Pseudostate))
                    /*si la transicion proviene de un pseudoestado
                    no se utiliza un segundo switch*/
                    {
                              println (t3+"switch ("+transicion.owner.name
                              +"Actual.getEstado()){")
                    }
                    transitionList->forEach(t: MetaModel.Transition)
                    {
                    sName = t.source.name
                    tName = t._getFeature("target").name
                    trans = t.name
//-------------------------------------------------------------------------------------
/*Se comprueba si el estado que lanza la transicion es un pseudoEstado,será del tipo JOIN,
ya que el pseudoEstado fork sólo recibe las transiciones lanzadas desde el simulador, pero
no las lanza*/
                    if((t.source.oclIsTypeOf(MetaModel.Pseudostate)) and
                    (t.source.kind = "JoinState"))
                    {
/*Si la transicion tiene como fuente un pseudoEstado join,antes de lanzar la transicion, se
debe comprobar si se encuentraen los estados necesarios para lanzar la transicion desde
JOIN*/
/*Para encontrar estos estados, primero se buscan las transiciones que tienen como destino,
el pseudoStado JOIN desde el que se ha lanzado la transicion*/
                              joinTrans = self.objectsOfType(MetaModel.Transition)-
                              >select(jt:MetaModel.Transition |
                    (jt._getFeature("target") == t.source))
                              print (t4+"if( ")
                              joinTrans->forEach(jt:MetaModel.Transition)
                              {
                                        if(position()!=0)print(" && ")
                                        print("("+jt.owner.name+"Actual ==
                                        "+jt.source.name+")")
          }
                              println (" ){")
                              joinTrans->forEach(jt:MetaModel.Transition)
```

```
                              {
                              if(jt.source.owner!=jt._getFeature("target").owner)
                                 {
/*Antes de comprobar el camino a seguir por cada transición,se debe salir del estado en el
que se encuentra, la actualización delpseudoEstado historico, si lo tiene, se realizará al
llamar el método comprobarLocalización()*/
                                println (t4 +"\t"+ jt.owner.name +
                              "Actual.onExit();");
                               comprobarLocalizacion(jt)
                                  }
                    }
/*Una vez se ha llegado al estado que contiene las transiciones que tienen como objetivo el
pseudoEstado Join, se sale de este estado y se lanza la transición*/
                              println (t4 +"\t"+ t._getFeature("target").owner.name +
                              "Actual.onExit();");
                              lanzarTransicion(t)
                              println (t4+" }")
                              } //fin if => la fuentes es un pseudoEstado de tipo
                              join
//-------------------------------------------------------------------------------------
         else      //la fuente es un estado
                    {
/*Se comprueba si el estado que lanza la transicion, contiene una región
 si es así, antes de lanzar la transición, debe salir de los estados interiores*/
                              println (t4+"case "+ sName + " : //Se encuentra en
                              el estado "+sName)
                              if(!t.source.region.isEmpty())
                               {
                              regiones = self.objectsOfType(MetaModel.Region)-
                              >select(reg:MetaModel.Region | (reg.owner ==
                    t.source))
                              regiones->forEach(rI:MetaModel.Region)
                                        {
                                        getEstadosInteriores(rI)-
                              >forEach(estInt:MetaModel.State)
                                          {
                              if(!estInt.region.isEmpty())
                              regiones.add(estInt.region)
                                          }
                                        }
                              while(!regiones.isEmpty())
                              {
                              println (t4 +"\t"+ regiones.last().name +
                    "Actual.onExit();");
/*Se comprueba si la region tiene historico, actualizándo su valor si existe*/
                              if(comprobarHistory(regiones.last()))
                              {
                                        println (t4 +"\t"+ regiones.last().name +
                              "History = "+regiones.last().name+"Actual;");
                              }
                              println (t4 +"\t"+ regiones.last().name + "Actual =
                              null;");
                              regiones.remove(regiones.last())
                              }
                    }
/* Esta a un mismo nivel de region, por lo que se sale del estado que lanza la transicion y
se lanza la transicion*/
                    println (t4 +"\t"+t.source.owner.name + "Actual.onExit();");
                    lanzarTransicion(t)
                    println (t4+"\tbreak; \n");
            }//else => la fuente es un estado
//-------------------------------------------------------------------------------------
/*Ya se ha comprobado las dos posibles fuentes que pueden lanzar transiciones. Las
siguientes líneas de código se encargarán de completar el código JAVA, añadiendo la clausula
default si la fuente no es un pseudoEstado, "else" si es un pseudoEstado y añadiendo la
forma de actuar si el objetivo es un estado de tipo final*/
//-------------------------------------------------------------------------------------
         if(!t.source.oclIsTypeOf(MetaModel.Pseudostate))
         {
                    println(t4+'default:')
                    println(t4 +'\t System.out.println ("No se puede realizar la
                    transicion '+trans+', se continua en estado: ");');
                    println(t4 + '\t getEstadosActualesToString();')
/*La siguiente excepcion sólo se lanzará si a la hora de crear el modelo, se elige la opción
UnexpectedTransPolicies => throwException*/
                    if(transPolicy = "throwException")
```

```
                    {
                              println(t4 +'\t throw new '+exceptionName+"();")
                    }
/*si la transicion proviene de un pseudoestado no se utiliza un segundo switch, por lo que
la siguiente llave no es usada*/
                    println (t3+"}");
                    }
//-------------------------------------------------------------------------------------/*Si la
transicion viene de un pseudo estado, al sustituir el segundo "switch", por
el condicional "if", en lugar  de la sentencia "default", se utiliza "else"*/
                    else
                    {
                              println(t4 +'else { ')
                              println(t4 +'\t System.out.println ("No se puede
                              realizar la transicion '+trans+', se continua
                              en estado: ");');
                              println(t4 +'\t getEstadosActualesToString();')
/*La siguiente excepcion sólo se lanzará si a la hora de crear el modelo, se elige la opción
UnexpectedTransPolicies => throwException*/
                              if(transPolicy = "throwException")
                              {
                                        println(t4 +'\t throw new
                                        '+exceptionName+"();")
                              }
                              println(t4+'}')
                    }
                    println ("\t\t break; \n");
//-------------------------------------------------------------------------------------
}//fin forEach(por cada transicion)
          }//fin if la lista no esta vacia
}  // writeProcedure


//-------------------------------------------------------------------------------------
module:: getEstadosInteriores (regionActual : MetaModel.Region):List  {
          var estadosInteriores : List
          estadosInteriores = self.objectsOfType(MetaModel.State)-
          >select(s:MetaModel.State | (s.owner == regionActual))
          result = estadosInteriores
}  // getEstadosInteriores
//---------------------------------------------------------------------------------------
module:: lanzarTransicion(trans2Fire : MetaModel.Transition): void
{
/*Declaracion de variables*/
          var tName: String = trans2Fire._getFeature("target").name
          var trans: String = trans2Fire.name
          var t4:String = "\t\t\t\t ";
          var forkTrans : List
          var nextState : MetaModel.State
          /*Condicionantes de lanzamiento*/
          /*Se comprueba si la transicion es externa*/
          if(trans2Fire.kind = "external")
          {
          /*Se comprueba si el destino es un pseudoEstado*/
             if(!trans2Fire._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
              {/*El destino es un estado*/
                   println (t4 +"\tfire."+trans+"_fire();");
                   comprobarLocalizacion(trans2Fire)
              }//fin el destino es un estado
//------------------------------------------------------------------
             else //el destino es un pseudoEstado
             {
/*En el caso de que el destino sea un pseudoEstado, este pseudoEstado ha de ser  de tipo
fork, ya que a los pseudoEstados iniciales no llegan transiciones y al  pseudoEstado join,
se llega automaticamente, una vez se llega a los estados necesarios*/
          /*CASO FORK*/
          if((trans2Fire._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
          and (trans2Fire._getFeature("target").kind = "ForkState"))
          {
          /*Si es de tipo fork, se recogen las transiciones que tienen como fuente este
          pseudoEstado y se pasa a los estados que se indique*/
```

```
        /*Al tratarse de un pseudoEstado fork, se ejecutarán varias transiciones, esto
        conlleva un cambio de región, ya que en una misma región no se puede estar en más
        de un estado */

        /*Antes de comprobar la localizacion, se realiza la entrada al estado al que se
        dirige, ya que esta entrada sólo debe realizarse una vez y no una    vez por cada
        transición*/

        println (t4 +"\tfire."+trans+"_fire();");

        forkTrans = self.objectsOfType(MetaModel.Transition)-
        >select(ft:MetaModel.Transition |
        (trans2Fire._getFeature("target")== ft.source))

         nextState = forkTrans.first()._getFeature("target")

         while(nextState.owner!=trans2Fire.source.owner)nextState=nextState.owner

         println (t4 +"\t"+ trans2Fire.source.owner.name +"Actual = " +
        nextState.name + ";");

         println (t4 +"\t"+ trans2Fire.source.owner.name + "Actual.onEntry();");

 /*Si el destino es un estado de tipo final, no ejecutará método ejecutar()*/


        if(!trans2Fire._getFeature("target").oclIsTypeOf(MetaModel.FinalState))
         println (t4 +"\t"+ trans2Fire.source.owner.name +
        "Actual.ejecutar();");

         forkTrans->forEach(ft:MetaModel.Transition)

            {

                println (t4 +"//----------------------------");

                comprobarLocalizacion(ft)

            }//fin forEach(cada transicion Fork)

          }//fin caso fork

        }//fin else, el destino es un pseudoEstado

      } //fin es una transicion externa

//-------------------------------------------------------------------------------------

/* transición interna*/

     else

        {

        println (t4 +"\tfire."+trans+"_fire();");

        println(t4 +"\t"+'System.out.println("          >>>> Estado actual de
        '+trans2Fire.owner.name+' : '+tName+'");')

        }

}
```

```
/-------------------------------------------------------------------------------------
module:: getPrimerEstadoName (region : MetaModel.Region) : String

{

         var primerEstadoName : String

         self.objectsOfType ( MetaModel.Transition )->forEach ( s :
         MetaModel.Transition )

         {

                  if((s.source.oclIsTypeOf(MetaModel.Pseudostate))and
                  ((s.source.kind =             "InitialState") or
         (s.source.kind = "HistoryState")) and
         (s.source.owner == region))

                  {

                           primerEstadoName = s._getFeature("target").name

                  }

         }

         result = primerEstadoName

}

//-------------------------------------------------------------------------------------

module:: comprobarLocalizacion(trans2Check : MetaModel.Transition): void

{

/*Este metodo se utiliza cada vez que se lance una transición y se encargará

de comprobar si la transicion, lleva a una region más interna o más externa y

actuar en consecuencia*/

/*Recibe como parametros, la transicion deseada*/

         var t4 = "\t\t\t\t"

         var fuenteList : List

           var destinoList : List

           var fuente : MetaModel.State

           var destino : MetaModel.State

           var regiones :  List

           var interior : boolean

/*Se rellenan las listas con todos los estados "propietarios" de la fuente y el destino*/

           fuente = trans2Check.source

           destino = trans2Check._getFeature("target")
```

```
 /*Se realiza a siguiente operación para introducir tb las transiciones entre estados que
esten en topRegion. Ya que en este caso, no entraría en el bucle while. */

 /*Se añaden los estados de los que se debe entrar y salir para realizar la transicion*/

            while((fuente.name != regionSM)or(destino.name!=regionSM))

            {

               if(fuenteList.last().name != regionSM)

               {

                  fuenteList.add(fuente)

                    if(fuente.owner.name == regionSM)

                                     fuenteList.add(fuente.owner)

                             fuente = fuente.owner

               }

               if(destinoList.last().name != regionSM)

                    {

                             destinoList.add(destino)

                             if(destino.owner.name == regionSM)

                                      destinoList.add(destino.owner)

                             destino = destino.owner

                    }

            }

/*Se han rellenado ambas listas hasta llegar a topRegion, pero habrá transiciones en las que
no se tiene que llegar a topRegion, por lo que se comparan ambas listas, y se borran las
entradas que coincidan, ya que serán estados comunes, de los que no tendrá que entrar ni
salir*/

                    interior = true

                    while(interior)

                    {

                             if(destinoList.last() == fuenteList.last())

                             {

                               destinoList.remove(destinoList.last())

                               fuenteList.remove(fuenteList.last())

                             }

                             else interior = false /*Cuando dejen de ser iguales,
                                      se sale del bucle*/

                             /*Si despues de borrar, alguna de las listas está
                             vacía, se sale del bucle*/

                             if(fuenteList.isEmpty() or
                             destinoList.isEmpty())interior = false

                    }

/*Una vez se han rellenado ambas listas, lo primero que se debe hacer antes de entrar en los
estados "propietarios" del objetivo, es salir de los estados "propietarios" de la fuente*/


/*El primer estado del que se debe salir es el estado fuente, que estará almacenado en las
primeras posiciones de la lista*/


/*Si la transición se dirige a un pseudoEstado de tipo join, se tendrá un estado común, del
que tendrán que salir todas las transiciones del pseudoEstado.Este estado será el que se
encuentre en la misma región que el destino y corresponde con el último introducido en la
lista, así que se borra y se sale de este estado cuando se vuela a la invocación de este
método, comprobarLocalizacion()*/

          if((trans2Check._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
          and(trans2Check._getFeature("target").kind == "JoinState"))

                             fuenteList.remove(fuenteList.last())


                    while(!fuenteList.isEmpty())

                    {

                    if(fuenteList.first().oclIsTypeOf(MetaModel.State))

                    {

/*Se comprueba si el estado del que se va a salir, en el recorrido de la transicion,
contiene una región si es así, antes de salir del estado, debe salir de los estados
interiores, a no ser que sea el estado que lanza la transición, ya que en este caso ya se ha
salido de sus estados interiores, antes de lanzar la transición*/

                                       if((fuenteList.first() !=
                                       trans2Check.source)and
                    (!fuenteList.first().region.isEmpty()))

                                       {

                                       regiones =
                             self.objectsOfType(MetaModel.Region)->
                    select(reg:MetaModel.Region | (reg.owner ==
          fuenteList.first()))

                                       regiones->forEach(rI:MetaModel.Region)

                                       {

                                                getEstadosInteriores(rI)-
                                       >forEach(estInt:MetaModel.State)

                                                {
```

```
                    if(!estInt.region.isEmpty())
                regiones.add(estInt.region)

                    }

/*En el caso de que el estado sea el estado fuente, ya se ha echo esta comprobacion y se ha
salido del estado antes de lanzar la transicion, por lo que se debe asegurar de que no
vuelva a salir de este estado*/

            regiones.remove(trans2Check.source.owner)

                }

                while(!regiones.isEmpty())

                  {

                    println (t4 +"\t"+
            regiones.last().name + "Actual.onExit();"); /*Se comprueba si
            la region tiene historico, actualizándo su valor si existe*/

                if(comprobarHistory(regiones.last()))

                    {

                println (t4 +"\t"+ regiones.last().name +
            "History = "+regiones.last().name+"Actual;");

                    }

                println (t4 +"\t"+ regiones.last().name +
            "Actual = null;");

                regiones.remove(regiones.last())

                  }

                }

/* Si la fuente que lanza la transición es de tipo State, antes de lanzar la transición ya
se ejecuta su función .onExit() antes de lanzar la transición, por lo que no se vuelve a
lanzar. Sí se establece su estado actual a null por tratarse de un cambio de región debido a
una salida*/

                if(fuenteList.first().owner !=
                trans2Check.source.owner)

                    println (t4 +"\t"+
                fuenteList.first().owner.name +
            "Actual.onExit();");

/*Sólo se debe poner a null los estados de las regiones de las que se sale*/

                if(fuenteList.first().owner !=
            trans2Check._getFeature("target").owner)

                {

/*Si la transición provoca una salida de la región,se comprueba si la region tiene
historico, actualizándo su valor si existe*/

        if(comprobarHistory(fuenteList.first().owner))

                    println (t4 +"\t"+ fuenteList.first().owner.name +
                "History = "+fuenteList.first().owner.name+ "Actual;");
        interior = false

                destinoList->forEach(dest : MetaModel.State)

                {

                    if(dest.owner == fuenteList.first().owner

                        interior = true
        }

/*Si la transicion se realiza a un estado interior de algun estado de la región en la que se
encuentra, el valor actual de la region será este estado, por lo que no se anctualiza a
null*/

                if(!interior)

                    println (t4 +"\t"+ fuenteList.first().owner.name +
                    "Actual = null;");

              }

/*Al salir de una region interior, se establece su valor de estado actual a null,

para que no permita transiciones erroneas. Tener en cuenta, que cuando se encuentra en la
misma región que el estado destino, no se sale de ella, se cambia su estado actual*/

          }

          fuenteList.remove(fuenteList.first())

        }//fin while

/*Una vez se ha salido de los estados necesarios, se va entrando a traves de los estados que
contienen el objetivo*/

/*Si la transición proviene un pseudoEstado de tipo fork, se tendrá un estado común, al que
tendrán que entrar todas las transiciones del pseudoEstado.Este estado será el que se
encuentre en la misma región que la fuente y corresponde con el ultimoo elemento de la
lista,así que se borra y se entra a este estado antes de la invocación de este método,
comprobarLocalizacion()*/


        if((trans2Check.source.oclIsTypeOf(MetaModel.Pseudostate))and(trans2Check.s
        ource.kind == "ForkState"))

                destinoList.remove(destinoList.last())

        while(!destinoList.isEmpty())

         {

        if((destinoList.last().oclIsTypeOf(MetaModel.State))or(destinoList.last().o
        clIsTypeOf(MetaModel.FinalState)))

         {

        if(destinoList.last().owner!=trans2Check._getFeature("target").owner)

                {
```

```
                              if(comprobarHistory(destinoList.last().owner))
                                        println (t4 +"\t"+
                                        destinoList.last().owner.name +
                              "Actual =
                    "+destinoList.last().owner.name+
          "History;");
                              else
                                        println (t4 +"\t"+  destinoList.last().owner.name   +
"Actual =                     "+getPrimerEstadoName(destinoList.last().owner)+ "Actual;");
                    }
                    else
                              println (t4 +"\t"+ destinoList.last().owner.name +
                              "Actual = "+destinoList.last().name+ ";");


                    println (t4 +"\t"+ destinoList.last().owner.name +
                    "Actual.onEntry();");
/*Si el destino es un estado de tipo final, no ejecutará método ejecutar()*/
          if(!destinoList.last().oclIsTypeOf(MetaModel.FinalState))
                              println (t4 +"\t"+ destinoList.last().owner.name +
                              "Actual.ejecutar();");
/*Se comprueba si el estado al que se pasa, contiene más regiones*/
          if(!destinoList.last().region.isEmpty())
          {
 /*Se recogen todas las regiones, cuyo propietario es el estado al que se ha entrado*/
                    regiones = self.objectsOfType(MetaModel.Region)-
                    >select(r:MetaModel.Region | (r.owner ==
          destinoList.last()))
                    regiones->forEach(regionActual:MetaModel.Region)
                    {
/*Como se ha entrado en una región, también se entra directamente al estado al que apunte su
pseudoEstado,por lo que se debe ejecutar sus funciones _onEntry() y _do()*/
                    if(comprobarHistory(regionActual))
                    {
/*Si la region tiene un pseudoEstado historico, al entrar a la region, se accede
al estado al que apunta el historico*/
                              println (t4 +"\t"+regionActual.name+"Actual =
                              "+regionActual.name+"History;");
                      }
                        else
                        {/*Si la region region no tiene historico, se accede al estado
                      apuntado por el pseudoEstado inicial */
                                  println (t4 +"\t"+regionActual.name+"Actual =
                                  "+getPrimerEstadoName(regionActual)+ ";");
                        }
                        println (t4 +"\t"+ regionActual.name + "Actual.onEntry();");
/*Si el estado al que se entra es un estado de tipo final, no ejecutará método ejecutar()*/
          if(!getPrimerEstadoName(regionActual).oclIsTypeOf(MetaModel.FinalState))
                    println (t4 +"\t"+ regionActual.name + "Actual.ejecutar();");
                    }//fin forEach regiones
            }//fin el estado tiene regiones interiores
          }
          destinoList.remove(destinoList.last())
      }//fin while(dl:MetaModel.State)
}
//-------------------------------------------------------------------------------------
module:: comprobarHistory (region2Check : MetaModel.Region):boolean  {
          var historico : boolean = false
          var pseudoEstados : List = self.objectsOfType(MetaModel.Pseudostate)
                    pseudoEstados->forEach(hist:MetaModel.Pseudostate)
          {
                    if((hist.kind == "HistoryState")and(hist.owner==region2Check))
                              historico = true
          }
          result = historico
}  // comprobarHistory
//-------------------------------------------------------------------------------------     //
texttransformation MOFScriptExample
```

B. 3 Código MOFSCRIPT. Patrón Estado.

```
/**
 * transformation SM_ModeltoJava_patronEstado
 */
texttransformation Ts2FOO (in MetaModel:StateMLplus) {
/*Se declaran estas variables globales, porque se usarán tanto en el main, como en los
métodos utilizados fuera de main*/
        var exceptionName : String = "transitionErrorException" //Nombre de la
        excepcion creada para transiciones erroneas
        var fatherException :String = "NullPointerException" //Nombre de la
        excepcion de JAVA de la cual hereda
        var regionSM : String = self.topRegion.name
        var transPolicy : String = self.unexpectedTransPolicy
    MetaModel.StateMachine::main(){
        var path : String = "StateMLplus/"+self.name+"/patronEstado"
        var package : String = "StateMLplus."+self.name+".patronEstado"
        var states : List = getStates ()
        var transitions : List = getTransitions()
        var regiones : List = self.objectsOfType(MetaModel.Region)
        var historyPseudo : List = self.objectsOfType(MetaModel.Pseudostate)-
        >select(ph:MetaModel.Pseudostate | (ph.kind =="HistoryState"))
        var joinTrans: List
        //--------------------------------------------
        //-------CLASE transitionErrorException--------
        //--------------------------------------------
        /*Esta clase sólo se creará si a la hora de crear el modelo, se elige la
        opción UnexpectedTransPolicies => throwException*/
        if(transPolicy = "throwException")
        {
                file excep(path+"\\"+exceptionName+".java");
                excep.println("package "+package+";");
                excep.println("public class "+exceptionName+ " extends
                "+fatherException+"{");
                excep.println("\t public "+exceptionName+"()\n \t {");
                excep.println('\t\t super("Lanzada excepción
                '+exceptionName+'");');
                excep.println("\t }");
                excep.println("}");
        }
        //--------------------------------------------
        //-----------interfaz finFSM------------------
        //--------------------------------------------
        /*Esta interfaz será implementada por FSM, e incluirá el método mediante el
        cuál un estado final notificará que se ha accedido a este tipo de estado.Esta
        proceso se llama proceso de CALLBACK*/
        file finInterf(path+"\\"+self.name+"_finFSMinterface.java");
        finInterf.println("package "+package+";");
        finInterf.println("public interface "+self.name+ "_finFSMinterface{");
        finInterf.println("\t public void notifyEnd();");
        finInterf.println("}");
        //--------------------------------------------
        //------INTERFAZ transitioInterface------------
        //--------------------------------------------
        file trs(path+"\\"+self.name+"_transitionInterface.java")
        trs.println("package "+package+";");
        trs.println("public interface "+self.name +"_transitionInterface{");
        trs.print("\t public enum transition { ")
        transitions -> forEach ( t : MetaModel.Transition )
        {
                if (position()!=0) trs.print (", ")
                        trs.print ( t.name )
        }
        trs.println("};\n")
        /*Funciones que invocan las transiciones*/
        trs.println("\n\t /*FUNCIONES INVOCADAS POR LAS TRANSICIONES*/")
        transitions -> forEach ( t : MetaModel.Transition )
        {
```

```
        trs.println("\t public void "+t.name+"();")
}
trs.println("}"); //FIN INTERFACE TRANSITION
//---------------------------------------------
//---------------CLASE FIRE--------------------
//---------------------------------------------
file fire(path+"\\"+self.name+"_fireTransition.java")
fire.println("package "+package+";");
fire.println("class "+self.name +"_fireTransition{");
fire.println("\n\t /*ESTA CLASE CONTIENE LAS IMPLEMENTACIONES DE LAS
FUNCIONES");
fire.println("\t DISPAROS DE LAS TRANSICIONES*/");
fire.println("\n\t /*EL USUARIO INSERTARÁ EN ESTAS FUNCIONES,");
fire.println("\t EL CÓDIGO DESEADO PARA CADA DISPARO*/");

/*Funciones que simulan el disparo de las transiciones*/
transitions->forEach(trans : MetaModel.Transition)
{
        fire.println("\t void "+trans.name+"_fire(){");
        fire.println ('\t\t System.out.println (" \t\t>>>> Ejecutando
        '+trans.name+'_fire...");')
        fire.println("\t}");
}
fire.println("}");
//---------------------------------------------
//---------------CLASE FSM---------------------
//---------------------------------------------
file fsm(path+"\\"+self.name+"_FSM.java")
fsm.println("package "+package+";");
fsm.println("public class "+self.name+"_FSM implements
"+self.name+"_transitionInterface {");
  /*Se crea UN OBJETO POR CADA POSIBLE ESTADO*/
fsm.println("\n\t /*Se crea un objeto por cada estado*/");
```

```
states->forEach(s:MetaModel.State)
{
        fsm.println("\t final "+self.name+"_"+s.name+"State "+s.name+";")
}
/*Ahora se evaluan los pseudoEstados, serán de tipo join*/
states -> forEach (p : MetaModel.Pseudostate)
        fsm.println("\t final "+self.name+"_"+p.name+"State "+p.name+";")

 /*se crean las variables que contendrán el estado actual de cada región*/
regiones->forEach(reg:MetaModel.Region)
{
        fsm.println("\t protected "+self.name+"_State
        "+reg.name+"Actual;");
}
 /*Variables para las regiones que tengan historico*/
historyPseudo->forEach(ph:MetaModel.Pseudostate)
{
        fsm.print("\t protected "+self.name+"_State
        "+ph.owner.name+"History;")
}
fsm.println("\n\t /*Se crea objeto de tipo finFSMinterface*/");
fsm.println("\t "+self.name+"_finFSMinterface user;")
fsm.println("\n\t /*SE CREA EL CONSTRUCTOR*/")
fsm.println("\n\t public "+self.name+"_FSM("+self.name+"_finFSMinterface
userNot){")
fsm.println("\n\t /*SE INICIALIZAN LOS ESTADOS*/")
states->forEach(s:MetaModel.State)
fsm.println("\t\t "+s.name+" = new "+self.name+"_"+s.name+"State(this);")
states->forEach(p:MetaModel.Pseudostate)
        fsm.println("\t\t "+p.name+" = new
        "+self.name+"_"+p.name+"State(this);")
fsm.println("\n\t /*Se inicializa topRegion y las regiones que tengan
historicos*/")
fsm.println("\t\t "+regiones.first().name+"Actual =
"+getPrimerEstadoName(regiones.first())+";")
```

```
/*Se crea una variable para cada region que tenga un pseudoEstado historico */

historyPseudo->forEach(ph:MetaModel.Pseudostate)

{

        fsm.print("\t\t "+ph.owner.name+"History =
        "+getPrimerEstadoName(ph.owner)+";")

}

fsm.println("\n\t/*Se inicializa el objeto que notificará el fin de la
maquina de estados*/")

fsm.println("\t\t user = userNot;");

fsm.println("\n\t }")//fin del constructor

fsm.println("\n\t/*Se crea un tipo de datos enumerado que contendrá los
estados actuales de cada región*/");

fsm.println("\t public enum region {");

regiones->forEach(reg:MetaModel.Region)

{

  /*Solo se inicializa topRegión, ya que será a la que se entre nada más
poner en funcionamiento la máquina de estados. Las demás variables se
inicializan a null*/

        if(position() ==0)

                fsm.print("\t\t
                "+reg.name+"("+self.name+
"_State.estado."+getPrimerEstadoName(reg)+")")

        if(position() !=0)

        {

                fsm.print(",\n")

                fsm.print("\t\t "+reg.name+"(null)")

        }

}

fsm.println(";")

fsm.println("\n\t\t public "+self.name+"_State.estado
currentState;/*Variable que almacenará el estado actual*/")

fsm.println("\n\t /*Se crea un constructor para inicializar la varible de e
stado actual, currentState*/")

fsm.println("\t\t region ("+self.name+"_State.estado estadoIni) {");

fsm.println("\t\t\t currentState = estadoIni;");

fsm.println("\t\t }");

fsm.println("\t }");

fsm.println("\n\t/*Se crea una instancia del enumerado region, para poder
acceder a sus campos*/");

fsm.println("\t/*Se deben inicializar a cualquier valor del enumerado, da
igual cual sea*/")

fsm.println("\t region estadoActual = region."+regiones.first().name+";")

/*FUNCION QUE DEVOLVERÁ EL ESTADO ACTUAL DE LA MÁQUINA DE ESTADOS.

DEVOLVERÁ UN TIPO DE DATOS ENUMERADO, QUE CONTENDRÁ EL ESTADO ACTUAL DE
CADA REGIÓN*/

fsm.println("\n\t /*FUNCION QUE DEVOLVERÁ EL ESTADO ACTUAL DE LA MÁQUINA DE
ESTADOS.");

fsm.println("\t DEVOLVERÁ UN TIPO DE DATOS ENUMERADO, QUE CONTENDRÁ EL
ESTADO ACTUAL DE CADA REGIÓN*/");

fsm.println("\t public region getEstadoActual (){");

fsm.println("\t\t return estadoActual;")

fsm.println("\t }");

/*FUNCION QUE SE ENCARGARÁ DE ACTUALIZAR EL VALOR DEL ENUMERADO QUE
CONTIENE LOS ESTADOS ACTUALES*/

fsm.println("\n\t/*FUNCION QUE SE ENCARGARÁ DE ACTUALIZAR EL VALOR DEL
ENUMERADO QUE CONTIENE LOS ESTADOS ACTUALES*/");

fsm.println("\t public void actualizarEstados(){")

regiones->forEach(reg:MetaModel.Region)

{

        fsm.println("\t\t if("+reg.name+"Actual != null)");

        fsm.println("\t\t\t estadoActual."+reg.name+".currentState =
        "+reg.name+"Actual.name;")

        fsm.println("\t\t else ")

        fsm.println("\t\t\t estadoActual."+reg.name+".currentState =
        null;")

}

fsm.println("\t }")

/*FUNCION QUE DEVOLVERÁ LOS ESTADOS ACTUALES DE TODAS LAS REGIONES EN
FORMATO STRING*/

fsm.println("\n\t /*FUNCION QUE DEVOLVERÁ LOS ESTADOS ACTUALES DE TODAS LAS
REGIONES*/");

fsm.println("\t public void getEstadosActualesToString(){")

regiones->forEach(reg:MetaModel.Region)
```

```
{
        fsm.println("\t\t\t\t if("+reg.name+"Actual != null)");
    fsm.println('\t\t\t\t\t System.out.println(">>>>>>>>>>
    Region '+reg.name+' en estado:"
+'+reg.name+'Actual.name);');

        fsm.println("\t\t\t\t else");

        fsm.println('\t\t\t\t\t System.out.println(">>>>>>>>>>
        Region '+reg.name+' no activa");');

        }

fsm.println("\t }")


/*CADA TRANSICIÓN POSIBLE, VENDRÁ IMPLEMENTADA COMO UNA FUNCIÓN*/

fsm.println("\n\t/*TRANSICIONES POSIBLES*/")

transitions->forEach(transicion : MetaModel.Transition)

{

    fsm.print("\t public void "+transicion.name+"()");

    if(transPolicy == "throwException")

    {

        fsm.print(" throws "+exceptionName)

        fsm.println(" {");

        fsm.println("\t\t try\n \t\t{");

    }

    if(transPolicy != "throwException")

        fsm.println("\t{")


    if(!transicion.source.oclIsTypeOf(MetaModel.Pseudostate))

    {

        fsm.println("\t\t\t
        if("+transicion.source.owner.name+"Actual ==
    "+transicion.source.name+"){")

        fsm.println("\t\t\t\t
        "+transicion.source.owner.name+"Actual."
    +transicion.name+"();")

        fsm.println("\t\t\t\t
        actualizarEstados();/*Se actualiza el
        enumerado que contiene los estados
        actuales*/")

        fsm.println("\t\t\t }")

    }

    if(transicion.source.oclIsTypeOf(MetaModel.Pseudostate))

    {

    /*Si la transicion tiene como fuente un pseudoStado, este será de
tipo Join, ya que los demás pseudoEstados no lanzan nsiciones*/

        joinTrans = self.objectsOfType(MetaModel.Transition)-
        >select(jt:MetaModel.Transition |
    (jt._getFeature("target") ==
transicion.source))

         fsm.print ("\t\t\t if( ")

         joinTrans->forEach(jt:MetaModel.Transition)

         {

            if(position()!=0)fsm.print(" && ")

            fsm.print("("+jt.owner.name+"Actual ==
        "+jt.source.name+")")
}

        fsm.println (" ){")

 /*El encargado de invocar la funcion asociada a la transición será el pseudoEstado join.
Como todos los elementos de la lista tienen como objetivo este pseudoEstado, se utiliza
cualquiera de ellos para acceder a este pseudoEstado*/

        fsm.println("\t\t\t\t
    "+joinTrans.first()._getFeature("target").name+
"."+transicion.name+"();")

        fsm.println("\t\t\t\t
            actualizarEstados();/*Se actualiza el
            enumerado que contiene los estados
            actuales*/")

        fsm.println("\t\t\t }")//fin if ( && )

    }

    fsm.println("\t\t\t else {")

    fsm.println('\t\t\t\t System.out.println("No se puede realizar la
        transicion '+transicion.name+', las regiones siguen en
        los estados: ");');

    fsm.println("\t\t\t\t getEstadosActualesToString ();")

/*La siguiente excepcion sólo se lanzará si a la hora de crear el modelo, se elige
la opción UnexpectedTransPolicies => throwException*/

    if(transPolicy = "throwException")

    {

        fsm.println('\t\t\t\t throw new '+exceptionName+"();")
```

```
          }
          fsm.println("\t\t\t }") //fin else
          if(transPolicy = "throwException")
          {
                    fsm.println("\t\t }//FIN TRY")//fin try
/*se captura la excepcion*/
                    fsm.println("/*se capturan excepciones de tipo
                    "+exceptionName+"*/")
                    fsm.println("\t\t catch ("+exceptionName+ " exc)\n\t\t
                    { ");
                    fsm.println('\t\t\t
                    System.out.println(exc.getMessage());')
                    fsm.println("\t\t }")
          }
          fsm.println("\t }")//fin public void
}//fin forEach(transicion : MetaModel.Transition)
/*Se crea un metodo a través del cual los estados finales, notificarán que se ha
entrado en uno de ellos.*/
fsm.println("\n\t/*EL SIGUIENTE METODO SE ENCARGA DE INFORMAR AL SIMULADOR
DE LA ENTRADA A") ;
fsm.println("\t UN ESTADO FINAL, POR LO QUE EL SIMULADOR DEBE
          FINALIZAR*/");
fsm.println("\n\t void notifyEnd( ){");
fsm.println('\t\t System.out.println ("           >>>> Entrada a estado
FINAL => FIN DEL SIMULADOR");');
fsm.println("\t\t user.notifyEnd();");
fsm.println("\t } \n"); //Fin notifyEnd()
fsm.println("}");//fin clase FSM
//---------------------------------------------
//---------------CLASE STATE-------------------
//---------------------------------------------
file claseStado(path+"\\"+self.name +"_State.java");
claseStado.println("package "+package+";");
claseStado.println("class "+self.name+ "_State{");
claseStado.println("\n\t /*SE RECOGEN LOS POSIBLES ESTADOS Y
TRANSICIONES*/");
claseStado.print("\t enum estado { ")
states -> forEach (s : MetaModel.State)
{
          if (position()!=0)
                    claseStado.print (", ")
          claseStado.print ( s.name )
}
/*Ahora se evalua los pseudoEstados(serán de tipo join)*/
states -> forEach (p : MetaModel.Pseudostate)
          claseStado.print (","+ p.name )
claseStado.println("};")
/*DECLARACION DE VARIABLES*/
claseStado.println("\n\t /*DECLARACION DE VARIABLES.");
claseStado.println("\t Se almacenará el nombre del estado y el nombre del
estado actual de cada region*/");
claseStado.println("\t estado name;");
/*Se utiliza un objeto de la clase FSM, para actualizar los estados  actuales   de
cada región*/
claseStado.println("\n\t/*Se utiliza un objeto de la clase FSM, para
actualizar los estados actuales de cada región*/");
claseStado.println("\t "+self.name+"_FSM theFSM;\n");
/*Se utiliza un objeto de la clase fireTransition, para simular el   disparo    de
las transiciones*/
claseStado.println("\n\t/*Se utiliza un objeto de la clase fire, para
simular el disparo de las transiciones*/");
claseStado.println("\t "+self.name+"_fireTransition fire = new
"+self.name+"_fireTransition();\n");
claseStado.println("\t estado getEstado(){");
claseStado.println("\t\t return name;");
claseStado.println("\t}");
/*Acciones de los estados*/
claseStado.println("\n\t /*ACCIONES DE LOS ESTADOS*/");
claseStado.println("\n\t /*No se declaran como funciones abstractas,porque
\n\t hay estados que no realizarán todas las acciones*/");
claseStado.println("\t void onEntry(){}");
```

```
claseStado.println("\t void ejecutar(){}");

claseStado.println("\t void onExit(){}");


/*Lanzamiento de transiciones por parte de los estados*/

claseStado.println("\n\t /*LANZAMIENTO DE TRANSICIONES POR PARTE DE LOS ESTADOS*/");

transitions -> forEach ( t : MetaModel.Transition )

{

          claseStado.println("\t void "+t.name+"(){}")

}

claseStado.println("}");//fin class State

//--------------------------------------------

//---------------CLASES DE CADA ESTADO---------

//--------------------------------------------

//SE CREA UNA CLASE POR CADA ESTADO

var name : String

var sourceTransitions : List //Almacena las transiciones que puede lanzar
                    un estado

states -> forEach ( s : MetaModel.State )

{

          name = s.name

          file name(path+"\\" + self.name + "_"+ name + "State.java")

          name.println("package "+package+";");

          name.println("class "+self.name+"_"+name+"State extends
          "+self.name+"_State{");


          name.println("/*Se crea el constructor*/")

          name.println("\t "+self.name+"_"+name+"State ("+self.name+"_FSM
          fsm){");

          name.println("\t\t name = estado."+name+";");

          name.println("\t\t theFSM = fsm;");

          name.println("\t }\n");

/*La función onEntry la implementarán todos los estados, pero los métodos
ejecutar() y onExit() sólo serán ejecutados por los estados que no sean de tipo
final*/
```

```
name.println("\t void onEntry(){");

name.println ("\t\t // -->>>> write code here...")

name.println ('\t\t System.out.println("\t\t>>>> Ejecutando ' +
s.name +'_onEntry...");')

if(s.oclIsTypeOf(MetaModel.FinalState))

          name.println('\t\t theFSM.notifyEnd();');

name.println("\t}");

if(!s.oclIsTypeOf(MetaModel.FinalState))

{

          name.println("\t void ejecutar(){");

          name.println ("\t\t // -->>>>> write code here...")

          name.println ('\t\t System.out.println ("\t\t>>>>
          Ejecutando '+ s.name +'_do...");')


          name.println ('\t\t System.out.println ("\t\t>>>>
          Estado actual de '+s.owner.name+' :
          '+s.name+'");')

          name.println("\t}");


          name.println("\t  void onExit(){");

          name.println ("\t\t // -->>>>> write code here...")

          name.println ('\t\t System.out.println ("\t\t>>>>
          Ejecutando ' + s.name +'_onExit...");')

          name.println("\t}");

}

sourceTransitions = transitions-
>select(transPropias:MetaModel.Transition |
(transPropias.source == s))

sourceTransitions->forEach(transicion:MetaModel.Transition)

{

/*Cada estado implementará sólo las transiciones que pueda
lanzar*/

          name.println("\t\t//----------------------------------
                    ")

          writeProcedure(transicion)

          name.println("\t\t//----------------------------------
                    ")
```

```
          }
          name.println("}");//fin forEach(s:MetaModel.State)
}
/*Ahora se evaluan los pseudoEstados, serán de tipo join*/
          states -> forEach (pj : MetaModel.Pseudostate)
          {
                    name = pj.name
                    file name(path+"\\" + self.name + "_"+ name +
                    "State.java")
                    name.println("package "+package+";");
                    name.println("class "+self.name+"_"+name+"State extends
                    "+self.name+"_State{");

                    name.println("/*Se crea el constructor*/")
                    name.println("\t "+self.name+"_"+name+"State
                    ("+self.name+"_FSM fsm){");
                    name.println("\t\t name = estado."+name+";");
                    name.println("\t\t theFSM = fsm;");
                    name.println("\t }\n");

                    sourceTransitions=transitions->
                              select(transPropias:MetaModel.Transition |
                              (transPropias.source == pj))
                    sourceTransitions-  > forEach(transicion:MetaModel.Transition)
                    {
                    /*Cada estado implementará sólo las transiciones que
                    pueda lanzar*/
                              name.println("\t\t//-------------------------
                    ---------")
                              writeProcedure(transicion)
                              name.println("\t\t//-------------------------
                    ---------")
                    }
          name.println("}");//fin pseudoEstado)
          }
// ---------------------------------------------
```

```
// ---------- simulator (main) ----------------
// --------------------------------------------
var opciones : integer;
file simu_adb (path+"\\"+self.name + "_Simulator.java")
simu_adb.println("package "+package+";");
simu_adb.println("public class " + self.name + "_Simulator implements
"+self.name+"_finFSMinterface{");
simu_adb.println("\t int opc;");
simu_adb.println("\t static boolean continuar;");
simu_adb.println("\t "+self.name +"_FSM FSM;")
simu_adb.println("\n\t /*SE CREA EL CONSTRUCTOR*/");
simu_adb.println("\t public "+self.name+"_Simulator(){");
simu_adb.println("\t\t FSM = new " + self.name+"_FSM(this);");
simu_adb.println("\t\t continuar = true;");
simu_adb.println("\t}");
simu_adb.println("\n\t/*Funcion que notificará el fin de la máquina de
estados*/")
simu_adb.println("\t public void notifyEnd(){");
simu_adb.println("\t\t continuar = false; //Para finalizar el bucle
infinito, se iguala su condición a false");
simu_adb.println("\t }");

simu_adb.println("\t public void fire_transition ( int opc ) {");
simu_adb.println("\t switch(opc){");
transitions -> forEach ( t : MetaModel.Transition ) {
          opciones = (count()+1)
          simu_adb.println("\t\t\t case " + opciones + " : FSM."+t.name+
          "();");
          simu_adb.println("\t\t\t\t\t break;");
}
opciones = opciones +1;
simu_adb.println("\t\t\t case "+opciones+" :
FSM.getEstadosActualesToString();")
simu_adb.println("\t\t\t\t\t break;");
```

```
simu_adb.println("\t\t\t default : break;");

simu_adb.println("\t\t } //Fin switch");

simu_adb.println("\t } // Fin fire_transition\n");

simu_adb.println("public static void main(String args[]){");

simu_adb.println("\t"+self.name+"_Simulator sm = new
"+self.name+"_Simulator();");

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

simu_adb.println('\t System.out.println (" Bienvenido a ' + self.name +
'_Simulator" );');

simu_adb.println("\t while(continuar){")

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

transitions -> forEach ( t : MetaModel.Transition ) {

          opciones = (count()+1)

          simu_adb.println('\t\t System.out.println ("    ' + opciones + ".-
" + t.name + '" );');

}

opciones = opciones +1;

simu_adb.println('\t\t System.out.println ("    ' + opciones + '.- Mostrar
estados actuales en pantalla");');

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

simu_adb.println('\t\t System.out.println ("    0.- Finalizar Simulador"
);');

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');
simu_adb.print('\t\t System.out.print ("   Selecciona transición a disparar
(0-' + opciones +'):  ");');

simu_adb.println("\n\t\t int elec = Teclado.readInt(  );");

simu_adb.println('\t System.out.println ("'+ '-----------------------------
-------------" );');

simu_adb.println('\t\t if (elec == 0){ \n\t\t\t System.out.println("Ha
elegido finalizar el simulador.");');

simu_adb.println("\t\t\t continuar = false; //Si opc=0, salir del bucle
infinito \n \t\t\t } ");

simu_adb.println("\t\t sm.fire_transition ( elec );");

simu_adb.println("\t } // Fin while")

simu_adb.println(" } // Fin main \n");

          simu_adb.println("} //Fin class");

} // module::main

//---------------------------------------------------------------------------------
module::getTransitions ( ) : List {

/*Este metodo se encargará de recoger las transiciones que no tengan como fuente o destino
un pseudoEstado, a no ser que el pseudoEstado fuente sea de tipo join o el pseudoEstado
destino sea del tipo fork */

          var transitions : List

          var repe : boolean

          var pseudoEstadoValido : boolean

          self.objectsOfType ( MetaModel.Transition )->forEach ( s :
          MetaModel.Transition )

           {

                    pseudoEstadoValido = false

                    if((s.source.oclIsTypeOf(MetaModel.Pseudostate)) and
                    (s.source.kind = "JoinState"))

                              pseudoEstadoValido = true

                     if((s._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
                    and (s._getFeature("target").kind = "ForkState"))

                              pseudoEstadoValido = true

                    if ((!s.source.oclIsTypeOf(MetaModel.Pseudostate))
          and(!s._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate)))
          pseudoEstadoValido = true

                    if(pseudoEstadoValido)

                    {

                              repe = false

                              transitions->forEach(c : MetaModel.Transition)

                              {

                                        if(c == s) repe = true

                              }

                              if(!repe){

                                        transitions.add ( s )

                              }

                    }

          }

          result = transitions
```

```
}  // getTransitionNames

//-----------------------------------------------------------------------------------
module::getTransitionNames ( ) : List {

         var transitionNames : List

         var transitions : List = getTransitions()

         transitions->forEach(t:MetaModel.Transition) transitionNames.add(t.name)

         result = transitionNames

}  // getTransitionNames

//-----------------------------------------------------------------------------------
module:: getStates ( ) : List {

         var states: List

         var repe : boolean

         self.objectsOfType ( MetaModel.State )->forEach ( s : MetaModel.State ) {

         repe = false

                  states->forEach(c:MetaModel.State)

                  {

                           if(c == s) repe = true

                  }

                  if(!repe){

                           states.add ( s )

                           if(!s.region.isEmpty())
                  {

                                    s.objectsOfType(MetaModel.State)->forEach(eI :
                           MetaModel.State)

                                    {

                                             states.add(eI)

                                    }

                           }

                  }

         }

/*Se añaden los pseudoEstados de tipo join, ya que estos tb pueden lanzar transiciones*/

         self.objectsOfType(MetaModel.Pseudostate)-
         >forEach(pj:MetaModel.Pseudostate)

                           if(pj.kind == "JoinState")states.add(pj)

         result = states

}  // getStates

//-----------------------------------------------------------------------------------
module:: writeProcedure (transicion : MetaModel.Transition)

{

         var sName:String  // source state name

         var tName:String  // target state name

         var trans:String // transition name

         var joinTrans : List

         var regiones : List

         println ("\t void " +transicion.name+ "(){")

         sName = transicion.source.name

         tName = transicion._getFeature("target").name

         trans = transicion.name

//-----------------------------------------------------------------------------------

/*Se comprueba si el estado que lanza la transicion es un pseudoEstado,será del tipo JOIN,
ya que el pseudoEstado fork sólo recibe las transiciones lanzadas desde el simulador, pero
no las lanza*/

         if((transicion.source.oclIsTypeOf(MetaModel.Pseudostate)) and
         (transicion.source.kind = "JoinState"))

         {

 /*Si la transicion tiene como fuente un pseudoEstado join,antes de lanzar la transicion, se
debe comprobar si se encuentra en los estados necesarios para lanzar la transicion desde
JOIN*/

/*Para encontrar estos estados, primero se buscan las transiciones que tienen como destino,
el pseudoStado JOIN desde el que se ha lanzado la transicion*/

                  joinTrans = self.objectsOfType(MetaModel.Transition)-
                  >select(jt:MetaModel.Transition |
         (jt._getFeature("target") == transicion.source))

                  joinTrans->forEach(jt:MetaModel.Transition)

                  {

                           if(jt.source.owner!=jt._getFeature("target").owner)

                           {

/*Antes de comprobar el camino a seguir por cada transición,se debe salir del estado en el
que se encuentra, la actualización del pseudoEstado historico, si lo tiene, se realizará al
llamar el método comprobarLocalización()*/

                                    println ("\t\t theFSM."+ jt.owner.name +
                                    "Actual.onExit();");
```

```
                                        comprobarLocalizacion(jt)
                              }
                    }
/*Una vez se ha llegado al estado que contiene las transiciones que tienen como objetivo el
pseudoEstado Join, se sale de este estado y se lanza la transición*/
          println ("\t\t theFSM."+ transicion._getFeature("target").owner.name +
          "Actual.onExit();");
          lanzarTransicion(transicion)
          } //fin if => la fuentes es un pseudoEstado de tipo join
//-------------------------------------------------------------------------------------
          else     //la fuente es un estado
           {
/*Se comprueba si el estado que lanza la transicion, contiene una región  si es así, antes
de lanzar la transición, debe salir de los estados interiores*/
                    if(!transicion.source.region.isEmpty())
                    {
                    regiones = self.objectsOfType(MetaModel.Region)-
                    >select(reg:MetaModel.Region | (reg.owner ==
          transicion.source))
                              regiones->forEach(rI:MetaModel.Region)
                              {
                                        getEstadosInteriores(rI)-
                                        >forEach(estInt:MetaModel.State)
                                        {

                    if(!estInt.region.isEmpty())regiones.add(estInt.region)
                                        }
                              }
                              while(!regiones.isEmpty())
                              {
                                        println ("\t\t theFSM."+ regiones.last().name
                                        + "Actual.onExit();");
/*Se comprueba si la region tiene historico, actualizándo su valor si existe*/
                                        if(comprobarHistory(regiones.last()))
                                        {
                                                  println ("\t\t theFSM."+
                                                  regiones.last().name +
                                        "History =
          theFSM."+regiones.last().name+"Actual;");
                                        }
                                        println ("\t\t theFSM."+ regiones.last().name
                                        + "Actual = null;");
                                        regiones.remove(regiones.last())
                              }
                    }
/*Ya se encuentr a un mismo nivel de region, por lo que se sale del estado que lanza la
transicion y se lanza la transicion*/
             println ("\t\t theFSM."+transicion.source.owner.name +
          "Actual.onExit();");
             lanzarTransicion(transicion)
           }//else => la fuente es un estado
//-------------------------------------------------------------------------------------
           println ("\t }");
//-------------------------------------------------------------------------------------
}  // writeProcedure
//-------------------------------------------------------------------------------------
module:: getEstadosInteriores (regionActual : MetaModel.Region):List  {
          var estadosInteriores : List
          estadosInteriores = self.objectsOfType(MetaModel.State)-
          >select(s:MetaModel.State | (s.owner == regionActual))
          result = estadosInteriores
}  // getEstadosInteriores
//-------------------------------------------------------------------------------------
module:: lanzarTransicion(trans2Fire : MetaModel.Transition): void
{
          /*Declaracion de variables*/
          var tName: String = trans2Fire._getFeature("target").name
          var trans: String = trans2Fire.name
          var forkTrans : List
          var nextState : MetaModel.State
```

```
/*Condicionantes de lanzamiento*/

/*Se comprueba si la transicion es externa*/

if(trans2Fire.kind = "external")

{

        /*Se comprueba si el destino es un pseudoEstado*/

        if(!trans2Fire._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))

    {/*El destino es un estado*/

        println ("\t\t fire."+trans+"_fire( );");

        comprobarLocalizacion(trans2Fire)

    }//fin el destino es un estado

//-------------------------------------------------------------------

else //el destino es un pseudoEstado

{

/*En el caso de que el destino sea un pseudoEstado, este pseudoEstado ha de ser

 de tipo fork, ya que a los pseudoEstados iniciales no llegan transiciones y al

 pseudoEstado join, se llega automaticamente, una vez se llega a los estados necesarios

*/

/*CASO FORK*/

        if((trans2Fire._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
        and (trans2Fire._getFeature("target").kind = "ForkState"))

{

/*Si es de tipo fork, se recogen las transiciones que tienen como fuente este pseudoEstado y
se pasa a los estados que se indique*/

/*Al tratarse de un pseudoEstado fork, se ejecutarán varias transiciones, esto conlleva un
cambio de región, ya que en una misma región se puede estar en más de un estado */

/*Antes de comprobar la localizacion, se realiza la entrada al estado al que se dirige, ya
que esta entrada sólo debe realizarse una vez y no una vez por cada transición*/

                     println ("\t\t fire."+trans+"_fire( );");

                     forkTrans = self.objectsOfType(MetaModel.Transition)-
                    >select(ft:MetaModel.Transition |
          (trans2Fire._getFeature("target")==
ft.source))

                    nextState = forkTrans.first()._getFeature("target")

while(nextState.owner!=trans2Fire.source.owner)

                    nextState=nextState.owner

                              println ("\t\t theFSM."+ trans2Fire.source.owner.name
                              +"Actual =theFSM." + nextState.name + ";");

                              println ("\t\t theFSM."+ trans2Fire.source.owner.name +
                              "Actual.onEntry();");
          println ("\t\t theFSM."+ trans2Fire.source.owner.name +
          "Actual.ejecutar();");

                              forkTrans->forEach(ft:MetaModel.Transition)

                              {

                                        println ("\t\t//----------------------------
                                        -");

                                        comprobarLocalizacion(ft)

                               }//fin forEach(cada transicion Fork)

                    }//fin caso fork

          }//fin else, el destino es un pseudoEstado

       } //fin es una transicion externa

//----------------------------------------------------------------------------------

                    /*transición interna*/

          else

          {

                    println ("\t\t fire."+trans+"_fire( );");

                    println("\t\t"+'System.out.println("             >>>> Estado
                    actual de '+trans2Fire.owner.name+' : '+tName+'");')

          }

}

//----------------------------------------------------------------------------------
module:: getPrimerEstadoName (region : MetaModel.Region) : String

{

          var primerEstadoName : String

          self.objectsOfType ( MetaModel.Transition )->forEach ( s :
          MetaModel.Transition )

          {

                    if((s.source.oclIsTypeOf(MetaModel.Pseudostate))and
                    ((s.source.kind = "InitialState") or (s.source.kind =
          "HistoryState")) and (s.source.owner == region))

                    {

                              primerEstadoName = s._getFeature("target").name

                     }
```

```
          }
          result = primerEstadoName
}
//-------------------------------------------------------------------------------------
module:: comprobarLocalizacion(trans2Check : MetaModel.Transition): void
{

/*Este metodo se utiliza cada vez que se lance una transición y se encargará de comprobar si
la transicion, conduce a una region más interna o más externa y actuar en consecuencia*/
  /*Recibe como parametros, la transicion deseada*/
          var fuenteList : List
          var destinoList : List
          var fuente : MetaModel.State
          var destino : MetaModel.State
          var regiones :  List
          var interior : boolean
 /*Se rellenan las listas con todos los estados "propietarios" de la fuente y el destino*/
          fuente = trans2Check.source
          destino = trans2Check._getFeature("target")
 /*Se realiza  la siguiente operación para introducir tb las transiciones entre estados que
esten en topRegion. Ya que en este caso, no entraría en el bucle while. */

/*Se añaden los estados de los que se debe entrar y salir para realizar la transicion*/
          while((fuente.name != regionSM)or(destino.name!=regionSM))
            {
               if(fuenteList.last().name != regionSM)
               {
                  fuenteList.add(fuente)
                    if(fuente.owner.name == regionSM)
                                        fuenteList.add(fuente.owner)
                              fuente = fuente.owner
               }
               if(destinoList.last().name != regionSM)
                    {
                              destinoList.add(destino)
                              if(destino.owner.name == regionSM)
                                        destinoList.add(destino.owner)
                              destino = destino.owner
                    }
            }
/*Se han rellenado ambas listas hasta llegar a topRegion, pero habrá transiciones en las que
no se tiene que llegar a topRegion, por lo que se comparan ambas listas, y se borran las
entradas que coincidan, ya que serán estados comunes, de los que no tendrá que entrar ni
salir*/
                    interior = true
                    while(interior)
                    {
                              if(destinoList.last() == fuenteList.last())
                              {
                                destinoList.remove(destinoList.last())
                                fuenteList.remove(fuenteList.last())
                              }
                              else interior = false /*Cuando dejen de ser iguales,
                                        se sale del bucle*/
/*Si despues de borrar, alguna de las listas está vacía, se sale del bucle*/
                              if(fuenteList.isEmpty() or destinoList.isEmpty())
                              interior = false
                    }
/*Una vez se ha rellenado ambas listas, lo primero que se debe hacer antes de entrar en los
estados "propietarios" del objetivo, es salir de los estados "propietarios" de la fuente*/
/*El primer estado del que se debe salir es el estado fuente, que estará almacenado en las
primeras posiciones de la lista*/
/*Si la transición se dirige a un pseudoEstado de tipo join, se tendrá un estado común, del
que tendrán que salir todas las transiciones del pseudoEstado.Este estado será el que se
encuentre en la misma región que el destino y corresponde con el último introducido en la
lista, así que se borra y se sale de este estado cuando vuelva a la invocación de este
método, comprobarLocalizacion()*/


          if((trans2Check._getFeature("target").oclIsTypeOf(MetaModel.Pseudostate))
          and(trans2Check._getFeature("target").kind == "JoinState"))
```

```
fuenteList.remove(fuenteList.last())

while(!fuenteList.isEmpty())

{

if(fuenteList.first().oclIsTypeOf(MetaModel.State))

{

/*Se comprueba si el estado del que se va a salir, en el recorrido de la transicion, contiene una región si es así, antes de salir del estado, debe salir de los estados interiores, a no ser que sea el estado que lanza la transición, ya que en este caso ya se ha salido de sus estados interiores, antes de lanzar la transición*/

if((fuenteList.first() !=
trans2Check.source)
and(!fuenteList.first().region.isEmpty()))

{

regiones =
self.objectsOfType(MetaModel.Region)-
>select(reg:MetaModel.Region | (reg.owner ==
fuenteList.first()))

regiones->forEach(rI:MetaModel.Region)

{

getEstadosInteriores(rI)-
>forEach(estInt:MetaModel.State)

{

if(!estInt.region.isEmpty())
regiones.add(estInt.region)

}

/*En el caso de que el estado sea el estado fuente, ya se ha echo esta comprobacion y se ha salido del estado antes de lanzar la transicion, por lo que se debe asegurar que no vuelva a salir de este estado*/
regiones.remove(trans2Check.source.owner)

}

while(!regiones.isEmpty())

{

println ("\t\t theFSM."+ regiones.last().name
+ "Actual.onExit();");

/*Se comprueba si la region tiene historico, actualizándo su valor si existe*/

if(comprobarHistory(regiones.last()))

{

println ("\t\t theFSM."+ regiones.last().name +
"History = theFSM."+regiones.last().name+"Actual;");

}

println ("\t\t theFSM."+ regiones.last().name + "Actual =
null;");

regiones.remove(regiones.last())

}

}

/* Si la fuente que lanza la transición es de tipo State, antes de lanzar la transición ya se ejecuta su función .onExit() antes de lanzar la transición, por lo que no se vuelve a lanzar. Sí se establece su estado actual a null por tratarse de un cambio de región debido a una salida*/

if(fuenteList.first().owner != trans2Check.source.owner)

println ("\t\t theFSM."+ fuenteList.first().owner.name +
"Actual.onExit();");

/*Sólo se deben poner a null los estados de las regiones de las que se sale*/

if(fuenteList.first().owner != trans2Check._getFeature("target").owner)

{

/*Si la transición provoca una salida de la región,se comprueba si la region tiene historico, actualizándo su valor si existe*/

if(comprobarHistory(fuenteList.first().owner))

{

println ("\t\t theFSM."+ fuenteList.first().owner.name
+ "History =
theFSM."+fuenteList.first().owner.name+
"Actual;");

}

interior = false

destinoList->forEach(dest : MetaModel.State)

{

if(dest.owner == fuenteList.first().owner)interior =
true

}

/*Si la transicion se realiza a un estado interior de algun estado de la region

en la que se encuentra, el valor actual de la region será este estado, por lo que no se actualiza a null*/

if(!interior)
```

```
                    {
                              println ("\t\t theFSM."+ fuenteList.first().owner.name
                              + "Actual = null;");
                    }
                }
/*Al salir de una region interior, se pone su valor de estado actual a null,  para que no
permita transiciones erroneas. Tener en cuenta, que cuando se encuentra en la misma región
que el estado destino, no se sale de ella, se cambia su estado actual*/

          }
          fuenteList.remove(fuenteList.first())
       }//fin while

/*Una vez se ha salido de los estados necesarios, se va entrando a traves de los estados que
contienen el objetivo*/
/*Si la transición proviene un pseudoEstado de tipo fork, se tendrá un estado común, al que
tendrán que entrar todas las transiciones del pseudoEstado.Este estado será el que se
encuentre en la misma región que la fuente y corresponde con el ultimoo elemento de la
lista,así que se borra y se entra a este estado antes de la invocación de este método,
comprobarLocalizacion()*/
          if((trans2Check.source.oclIsTypeOf(MetaModel.Pseudostate))and(trans2Check.s
          ource.kind == "ForkState"))
                    destinoList.remove(destinoList.last())

            while(!destinoList.isEmpty())
            {
                    if((destinoList.last().oclIsTypeOf(MetaModel.State))
          or(destinoList.last().oclIsTypeOf(MetaModel.FinalState)))
                    {
                    println ("\t\t theFSM."+ destinoList.last().owner.name + "Actual
                    = theFSM."+destinoList.last().name+ ";");
                    println ("\t\t theFSM."+ destinoList.last().owner.name +
                    "Actual.onEntry();");
                    println ("\t\t theFSM."+ destinoList.last().owner.name +
                    "Actual.ejecutar();");
/*Si es el último elemento de la lista, es porque se ha llegado al destino, asi que se
comprueba si este estado contiene regiones interiores, para entrar a su estado inicial*/
                              if(destinoList.last()==destinoList.first())
                              {
            /*Se comprueba si el estado al que se pasa, contiene más regiones*/
                              if(!destinoList.last().region.isEmpty())
                              {
/*Se recogen todas las regiones, cuyo propietario es el estado al que se ha entrado*/
                              regiones = self.objectsOfType(MetaModel.Region)-
                              >select(r:MetaModel.Region | (r.owner ==
                    destinoList.last()))
                               regiones->forEach(regionActual:MetaModel.Region)
                                {
 /*Como se ha entrado en una región, también se netra directamente al estado al que apunte
su pseudoEstado,por lo que se debe ejecutar sus funciones _onEntry() y _do()*/
                              if(comprobarHistory(regionActual))
                                {
/*Si la region tiene un pseudoEstado historico, al entrar a la region, se accede al estado
al que apunta el historico*/
                              println ("\t\t theFSM."+regionActual.name+"Actual =
                              theFSM."+regionActual.name+"History;");
                                }
                                else
                               {/*Si la region region no tiene historico, se accede
                    al estado apuntado por el pseudoEstado inicial */
                                        println ("\t\t theFSM."+
                              regionActual.name+"Actual =
                    theFSM."+getPrimerEstadoName(regionActual)+
          ";");
                                }
                                println ("\t\t theFSM."+ regionActual.name +
                              "Actual.onEntry();");
                                        println ("\t\t theFSM."+ regionActual.name +
                              "Actual.ejecutar();");
                              }//fin forEach regiones
                             }//fin el estado tiene regiones interiores
                    }
                }
          destinoList.remove(destinoList.last())
          }//fin while(dl:MetaModel.State)
}
```

```
//------------------------------------------------------------------------------------
module:: comprobarHistory (region2Check : MetaModel.Region):boolean  {

         var historico : boolean = false

         var pseudoEstados : List = self.objectsOfType(MetaModel.Pseudostate)

                  pseudoEstados->forEach(hist:MetaModel.Pseudostate)

                  {

                  if((hist.kind == "HistoryState")and(hist.owner==region2Check))

                           historico = true

                  }

         result = historico

}  // comprobarHistory

//------------------------------------------------------------------------------------

} // texttransformation MOFScriptExample
```

B. 4 Ejemplo de simulación

```
------------------------------------------
 Bienvenido a SMExample3_Simulator
------------------------------------------
   1.- go
   2.- contCmd
   3.- error
   4.- done
   5.- abort
   6.- sysRecovered
   7.- end
   8.- reStart
   9.- initialized
   10.- oriented
   11.- armInitialized
   12.- calcDone
   13.- movementDetect
   14.- Mostrar estados actuales en pantalla
------------------------------------------
   0.- Finalizar Simulador
------------------------------------------
   Selecciona transición a disparar (0-14):  2
------------------------------------------
                    >>>> Ejecutando Ready_onExit...
                    >>>> Ejecutando contCmd_fire...
                    >>>> Ejecutando Moving_onEntry...
                    >>>> Ejecutando Moving_do...
                    >>>> Estado actual de topRegion : Moving
                    >>>> Ejecutando OrientedTool_onEntry...
                    >>>> Ejecutando OrientedTool_do...
                    >>>> Estado actual de toolRegion : OrientedTool
                    >>>> Ejecutando CalculatingPath_onEntry...
                    >>>> Ejecutando CalculatingPath_do...
                    >>>> Estado actual de armRegion : CalculatingPath
------------------------------------------
   1.- go
   2.- contCmd
   3.- error
   4.- done
   5.- abort
   6.- sysRecovered
   7.- end
   8.- reStart
   9.- initialized
   10.- oriented
   11.- armInitialized
   12.- calcDone
   13.- movementDetect
   14.- Mostrar estados actuales en pantalla
------------------------------------------
   0.- Finalizar Simulador
------------------------------------------
   Selecciona transición a disparar (0-14):  10
------------------------------------------
                    >>>> Ejecutando OrientedTool_onExit...
                    >>>> Ejecutando oriented_fire...
                    >>>> Ejecutando PositionTool_onEntry...
                    >>>> Ejecutando PositionTool_do...
                    >>>> Estado actual de toolRegion : PositionTool
------------------------------------------
   1.- go
   2.- contCmd
   3.- error
   4.- done
   5.- abort
   6.- sysRecovered
   7.- end
   8.- reStart
   9.- initialized
   10.- oriented
   11.- armInitialized
   12.- calcDone
   13.- movementDetect
   14.- Mostrar estados actuales en pantalla
------------------------------------------
   0.- Finalizar Simulador
------------------------------------------
   Selecciona transición a disparar (0-14):  12
------------------------------------------
                    >>>> Ejecutando CalculatingPath_onExit...
                    >>>> Ejecutando calcDone_fire...
                    >>>> Ejecutando MovingArm_onEntry...
                    >>>> Ejecutando MovingArm_do...
                    >>>> Estado actual de armRegion : MovingArm
------------------------------------------
   1.- go
   2.- contCmd
   3.- error
   4.- done
   5.- abort
   6.- sysRecovered
   7.- end
   8.- reStart
   9.- initialized
   10.- oriented
   11.- armInitialized
   12.- calcDone
   13.- movementDetect
   14.- Mostrar estados actuales en pantalla
------------------------------------------
   0.- Finalizar Simulador
------------------------------------------
   Selecciona transición a disparar (0-14):  14
------------------------------------------
>>>>>>>>>> Region topRegion en estado:Moving
>>>>>>>>>> Region toolRegion en estado:PositionTool
>>>>>>>>>> Region armRegion en estado:MovingArm
------------------------------------------
   1.- go
   2.- contCmd
   3.- error
   4.- done
   5.- abort
   6.- sysRecovered
   7.- end
   8.- reStart
   9.- initialized
```

```
   10.- oriented
   11.- armInitialized
   12.- calcDone
   13.- movementDetect
   14.- Mostrar estados actuales en pantalla
------------------------------------------
   0.- Finalizar Simulador
------------------------------------------
   Selecciona transición a disparar (0-14):  4
------------------------------------------
                   >>>> Ejecutando PositionTool_onExit...
                   >>>> Ejecutando MovingArm_onExit...
                   >>>> Ejecutando Moving_onExit...
                   >>>> Ejecutando done_fire...
                   >>>> Ejecutando Finished_onEntry...
                   >>>> Ejecutando Finished_do...
                   >>>> Estado actual de topRegion : Finished
------------------------------------------
   1.- go
   2.- contCmd
   3.- error
   4.- done
   5.- abort
   6.- sysRecovered
   7.- end
   8.- reStart
   9.- initialized
   10.- oriented
   11.- armInitialized
   12.- calcDone
   13.- movementDetect
   14.- Mostrar estados actuales en pantalla
------------------------------------------
   0.- Finalizar Simulador
------------------------------------------
   Selecciona transición a disparar (0-14):  7
------------------------------------------
                   >>>> Ejecutando Finished_onExit...
                   >>>> Ejecutando end_fire...
                   >>>> Ejecutando End_onEntry...

                   >>>> Entrada a estado FINAL => FIN DEL SIMULADOR
```

7. BIBLIOGRAFÍA

1 David Harel, "StateCharts: A visual formalism for complex systems", 1984

2 P.K. Winston, "Artificial Intelligence", Addison-Wesley, 1993

3 The Unified Modeling Language Superstructure version 2.0.,OMG Final Adopted Specification. April 2004. http://www.omg.org.

4 Bézivin, J. MDA: "From Hype to Hope, and Reality"

5 Schmidt, D.C., "Model-Driven Engineering", IEEE Computer Society, Febrero (2006), pp. 25 –31.

6 Dragan Gasevic, Dragan Djuric, Vladan Devedzic, and Bran Selic, "Model Driven Architecture and Ontology Development", 2006

7 OMG: MDA Guide Version 1.0.1, OMG, 2003.

8 The Unified Modeling Language Infrastructure version 2.0, OMG Final Adopted Specification. March 2005. http://www.omg.org

9 Erich Gamma & Kent Beck, "Contributing to Eclipse: Principles, Patterns, and Plug-Ins", 2004

10 MOFScript - Project plan for a Eclipse GMT subproject, 21/12/2005.

11 Dr. Harvey Deitel & Paul Deitel, "JAVA How to Program" 6th edition, 2005

12 Freeman Eric. "Head First Desing Patterns / Eric Freeman, Elisabeth Freeman, with Kathy Sierra, Bert Bates". Sebastopol (California) : O'Reilly, 2004.

13 Frank Budinsky, Dave Steinberg, Ed Merks, Ray Ellersick, Timothy J. Grose, "Eclipse Modeling FrameWork", 2003.