Instituto de Computación
Facultad de Ingeniería
Universidad de la República

FRAMEWORK DE SIMULACIÓN BASADO EN SISTEMAS MULTI-AGENTE

Informe de Proyecto de Grado presentado al Tribunal de Evaluación como requisito de graduación de la carrera Ingeniería en Computación, Universidad de la República

2009 - 2010
Montevideo – Uruguay

Autores:

Diego Guerra
Paola Iraola
Yanella Sánchez

Supervisores:

Jorge Corral
Daniel Calegari

Resumen

Los Sistemas Multi-Agente (SMA) resultan ser una herramienta adecuada para modelar la interacción entre varios actores (agentes), su comportamiento y los recursos que estos consumen o producen. Este enfoque es beneficioso en situaciones en las que, debido a la multiplicidad y heterogeneidad de actores y su iteración en el tiempo, no es posible, o no es recomendable, una resolución algorítmica. Existen líneas de trabajo que combinan el modelado de una realidad y la construcción de SMA para luego realizar simulaciones de escenarios futuros.

El propósito de este proyecto es el estudio de factibilidad del desarrollo de un framework de simulación basado en SMA que mejore las prestaciones de los entornos de simulación disponibles en la actualidad. El framework deberá permitir el uso de un lenguaje específico de dominio para la especificación de un SMA y la construcción semiautomática del código de la simulación.

Como parte del trabajo se realizó una búsqueda exhaustiva de los frameworks existentes basados en SMA. Debido a que no se encontró un framework que se adecuara completamente a los criterios requeridos, se optó por la extensión de aquel que en mayor medida se aproximaba a nuestras necesidades: Ascape. Mediante la construcción de un prototipo, se pudo evaluar la factibilidad del desarrollo del framework, cuya implementación se basa en la interoperabilidad de las tecnologías EMF, GMF, Acceleo y del framework ya existente Ascape, integrados en la plataforma Eclipse.

Índice general

Capítulo 1

Introducción

Los Sistemas Multi-Agente [01] (SMA) permiten modelar situaciones en las que, debido a la multiplicidad y heterogeneidad de actores (agentes) y su iteración en el tiempo, no es posible, o no es recomendable, una resolución algorítmica. Por tanto, se construye un sistema que modela el comportamiento de estos agentes heterogéneos, los recursos que estos consumen o producen, así como la interrelación agente-agente y agente-recurso.

Actualmente existe una línea de trabajo que combina el modelado de una realidad y la construcción de SMA para luego realizar simulaciones de escenarios futuros. Esta línea es particularmente utilizada para simulación de agro-ecosistemas. Un agro-ecosistema es un ecosistema sujeto a la alteración y manipulación humana con el fin de establecer la producción agrícola.

En este contexto, la realidad es modelada en base a SMA, especificando diagramas UML (Unified Modeling Language) [02], para la especificación de la estructura del sistema (diagramas de clases) y para el comportamiento del mismo (diagramas de actividad), para luego desarrollar una simulación con la información generada utilizando un framework de simulación para SMA como por ejemplo Cormas [03].
Debido a que los diagramas utilizados no son lo suficientemente expresivos como para contemplar la particularidad del enfoque, se utiliza un lenguaje específico de dominio (lenguaje que ofrece mayor expresividad dentro de un dominio particular) para el modelado de las simulaciones basadas en SMA, lo que facilita el proceso de especificación del SMA y la posterior generación de código. No obstante, el pasaje de la especificación a la simulación se realiza a mano ya que no existe integración alguna entre la herramienta de modelado y el framework de simulación.

El objetivo general de este proyecto es el estudio de factibilidad del desarrollo (ya sea creando uno nuevo o extendiendo uno existente) de un framework de simulación basado en SMA, que mejore las capacidades de los entornos de simulación existentes en la actualidad. En lo que sigue de este documento, al framework a desarrollar lo llamaremos FW4SimSMA (ya que el framework es para simulaciones basado en SMA).
Es deseable que el framework cuente con las capacidades que se detallan a continuación:

- Esté integrado a Eclipse [04] (programado en Java [05]) de manera tal de facilitar su portabilidad y extensibilidad, teniendo en cuenta que Eclipse es un IDE para Java muy utilizado, que permite el uso de plugins como mecanismo de extensión.
- Permita el uso de un lenguaje específico de dominio [06] (DSL) para la especificación de un SMA y su posterior simulación.
- Permita la construcción semiautomática de código Java para la ejecución de una simulación de manera integrada al framework a partir de un DSL.
- La licencia del framework esté disponible de forma gratuita.

- En caso de tomarse como base para la extensión un framework existente, es necesario que presente una completa documentación, esté actualizado y cuente con una comunidad activa de usuarios.

Los objetivos concretos del proyecto son:

- Estudio y relevamiento de los distintos frameworks existentes de Simulación basados en SMA con el propósito de realizar un análisis comparativo de los mismos, para luego identificar aquellos que puedan ser tomados como base para la extensión o creación de un nuevo framework.

- Desarrollo de un framework integrado a Eclipse, que permita el uso de un DSL para especificar un SMA, para la posterior construcción y ejecución de una simulación. FW4SimSMA tendrá carácter de prototipo, ya que lo que se pretende es evaluar la factibilidad de su desarrollo.

- Validación del FW4SimSMA mediante la incorporación de un DSL para la simulación de agro-ecosistemas, la incorporación de un framework (ya existente) para la generación de una simulación a partir de dicho lenguaje y la aplicación y evaluación del mismo a partir de la implementación de un caso de estudio.

- Evaluación de los resultados obtenidos.

El resto del documento se organiza de la siguiente manera: El capítulo 2 presenta un conjunto de conceptos básicos necesarios para contextualizar el problema y la solución. En el capítulo 3 se resume el relevamiento realizado sobre frameworks de simulación basados en SMA. En el capítulo 4 se brinda información sobre la construcción del FW4SimSMA, junto a las decisiones tomadas en dicho desarrollo.
En el capítulo 5 se presenta el caso de estudio implementado para la evaluación del FW4SimSMA basado en agro-ecosistemas. Finalmente, en el capítulo 6 se resume el trabajo realizado y se presenta una reseña de posibles trabajos a futuros.

Capítulo 2

Contexto

En este capítulo se describen los conceptos básicos que formaron parte del estudio preliminar del contexto, como lo son SMA, simulación basada en SMA, Agro-Ecosistema y Modelos basados en Agentes; tomando como referencias a [01], [07], [08], [09].

2.1. Sistemas Multi-Agente

Existen distintas definiciones sobre el concepto de agente. Una ampliamente difundida expresa en términos generales lo siguiente:
Un **agente** es un sistema que está en algún ambiente y que es capaz de tomar acciones autónomas de acuerdo al estado del ambiente, para cumplir sus objetivos de diseño.
Un agente tiene por lo menos las siguientes características:

- La autonomía, es decir, un control sobre su propio estado y el comportamiento sobre la base de estímulos externos o internos.
- La capacidad para interactuar con su medio ambiente, incluyendo las percepciones y las acciones a efectuar.

El **ambiente** es lo que rodea al agente, es un entorno físico o virtual que proporciona las condiciones necesarias en donde los agentes existen y funcionan. En un contexto de SMA el ambiente también permite a los agentes comunicarse entre sí.

Es fundamental la relación entre el agente y el ambiente, en base a ello podemos clasificar el ambiente en:

- **Accesible vs Inaccesible**: Es accesible si el agente puede obtener información completa, precisa y actualizada sobre el ambiente, de lo contrario el ambiente es inaccesible.
- **Determinista vs No Determinista**: Un ambiente determinista es aquel en el que toda acción tiene un único efecto garantizado, no hay incertidumbre sobre el estado que será el resultado de ejecutar una acción, en caso contrario es no determinista.
- **Estático vs Dinámico**: Un ambiente estático es aquel que se puede suponer que no variará, salvo por el desempeño de las acciones de los agentes. Un ambiente dinámico tiene otros procesos que operan sobre él y por lo tanto, cambios en los modos más allá de controlar el agente.
- **Discreto vs Continuo**: Un ambiente es discreto si hay un número fijo y finito de acciones y percepciones en el mismo, de lo contrario se dice que estamos en un ambiente continuo.

Un **recurso** es una entidad física o virtual, cuya importancia radica principalmente en su disponibilidad (afectada por el consumo o producción). En el contexto de agro-ecosistemas nos referiremos a recursos naturales, los cuales clasificaremos en renovables y no renovables. Los recursos renovables son aquellos recursos cuya existencia no se agota con su utilización, debido a que vuelven a su estado original o se regeneran a una tasa mayor a la tasa con que los recursos son disminuidos mediante su

utilización, mientras que los recursos no renovables son recursos naturales que no pueden ser producidos, cultivados, regenerados o reutilizados a una escala tal que pueda sostener su tasa de consumo.

Un **Sistema Multi-Agente**, es un sistema compuesto por múltiples agentes, que conviven en un ambiente y que interactúan entre sí para cumplir sus objetivos individuales.

2.2. Simulación basada en sistemas Multi-Agente

Una **simulación** [10], [11] puede verse como la imitación de un proceso o sistema del mundo real a través del tiempo, con la finalidad de comprender su comportamiento y/o evaluar nuevas posibilidades.

Tipos de simulación:

- **Simulación Discreta**: El estado de las variables cambian instantáneamente en instantes de tiempo separados.
- **Simulación Continua**: El estado de las variables cambian continuamente en el tiempo. Es decir el paso del tiempo puede ser reducido a intervalos arbitrariamente pequeños.
- **Time-stepped** es un caso particular de simulación discreta, donde el modelo se ejecuta cada cierta cantidad fija de tiempo.

La simulación basada en SMA [12], [13], [14] es recomendable cuando se trata de simular el funcionamiento de sistemas que están compuestos por elementos heterogéneos que interactúan y donde su ubicación en el espacio resulta ser relevante.

Para la simulación basada en SMA se utilizan frameworks [15] que permiten el desarrollo de aplicaciones basadas en éste enfoque. Particularmente, nos basaremos en el uso de frameworks de simulación basados en SMA, tomando a los agro-ecosistemas como área de aplicación.

2.3. Agro-Ecosistemas

Cuando existen múltiples componentes con dinámicas propias y que interactúan entre sí, el modelado y simulación con SMA se propone como una herramienta especialmente adecuada para explorar la evolución del sistema en su conjunto y de cada uno de sus componentes [16]. Debido a esto es que nos centraremos en el modelado y simulación de agro-ecosistemas, como aplicación de SMA.

Un **agro-ecosistema** [17] es un ecosistema sujeto a la alteración y manipulación humana con el fin de establecer la producción agrícola**.** Se puede ver como un ecosistema sometido por el hombre a continuas modificaciones de sus componentes. Es un proceso generador de cambios intensos, que resulta de la interacción entre características (biológicas y ambientales) y factores (sociales y económicos).

2.4. Modelo Basado en Agentes

Un **modelo basado en agentes** [18] (ABM por sus siglas en inglés) es un tipo de modelo computacional que permite la representación y posterior simulación de acciones e interacciones de individuos autónomos dentro de un ambiente.
Un ABM es una herramienta para el estudio de sistemas (por ej.: sistemas sociales). Mediante la simulación de un ABM, se puede evaluar de manera sistemática diferentes hipótesis relativas a las propiedades de los agentes, sus reglas de comportamiento, los tipos de interacciones y su efecto a nivel macro de los hechos modelados del sistema.

En el presente proyecto, para la representación del ABM se utilizan diagramas, los cuales se basan en AML (Agent Modeling Lenguje) [19] y UML. AML es un lenguaje semi-formal para el modelado visual, que permite especificar modelos y documentación de los sistemas que incorporan conceptos de SMA.
Debido a que los diagramas UML no son lo suficientemente expresivos para contemplar las particularidades correspondientes a un SMA, se utiliza un DSL, el cual ofrece mayor expresividad dentro de un dominio particular (en este caso agro-ecosistemas).

2.5. Ejemplo SMA

A través del ejemplo de la propagación del virus de gripe A (N1H1), se modela y ejemplifica el uso de SMA.
Supongamos que un país tiene un porcentaje de su población sana y otro infectada por la gripe A. Se determinó que el virus se propaga entre la población por el contacto directo con una persona enferma y que la única forma que una persona sana no se contagie es porque se suministró (consumió) una vacuna trivalente (las cuales son agotables). Cabe destacar que el objetivo que persiguen las personas sanas es evitar el contagio y las personas enfermas es curarse, para que la enfermedad no derive en una pandemia.
Si la situación anterior la modelamos como un SMA, la población sana representa a un tipo de agente y la población enferma representa a otro. Las vacunas representan a recursos cuya disponibilidad se ve afectada por la cantidad de personas a las que se les suministra y el país representa el ambiente en donde las personas (agentes) viven e interactúan. En el contexto ejemplificado, el SMA está compuesto por toda la población (sana y enferma) del país el cual proporciona las condiciones necesarias para que las personas interactúen persiguiendo cada una de ellas su objetivo (curarse o evitar la enfermedad). Mediante un SMA se podría realizar el seguimiento de la propagación del virus (por ej. porcentaje de personas contagiadas), así como el monitoreo de los diferentes factores (por ej. cantidad de vacunas disponibles) que afectan a la población.
La Figura 2.1, ilustra el estado en un instante de tiempo, de los diferentes agentes ejemplificados, donde las caras felices representan a las personas sanas, las caras tristes a las personas infectadas por el virus y los cuadrados grises representan las distintas ubicaciones dentro del país (ambiente) que tienen las personas antes detalladas.

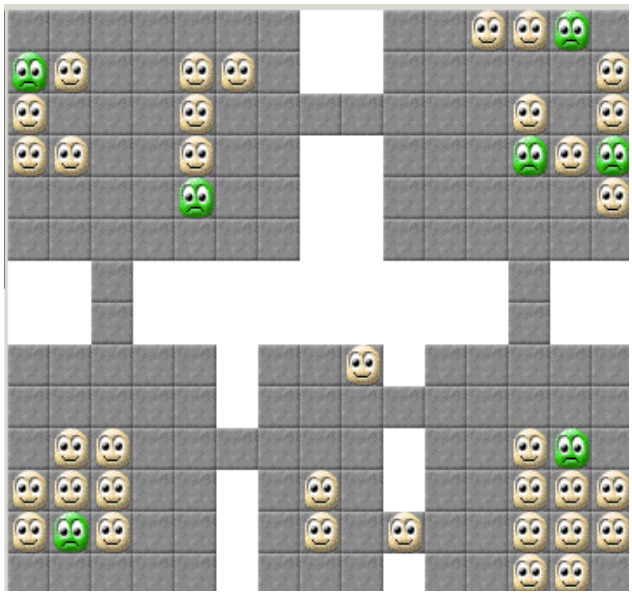

Figura 2.1: Ejemplo de SMA.

Capítulo 3

Frameworks de Simulación basados en Sistemas Multi-Agente

Se investigó y evaluó un conjunto existente de frameworks de simulación basados en SMA. Esta investigación surge de la necesidad de tener un framework que pueda ser tomado como base para la extensión o creación de uno nuevo, que proporcione funcionalidades y características específicas [20], referidas principalmente al licenciamiento, a su integración con Eclipse, que permita incorporar aspectos específicos de un dominio a nivel del modelado y simulación, que permita además, la construcción semiautomática de código para la realización de una simulación de manera integrada al framework y presente la mayor documentación posible contando con una comunidad de usuarios activa.

Se comenzó con la investigación de Cormas por ser el framework sugerido por los tutores del presente proyecto y utilizado en proyectos previos [21], los cuales motivaron el origen de este trabajo.
Cormas es un framework para simulaciones Multi-Agente, que está enfocado en la administración de recursos naturales renovables. Ofrece un conjunto de clases (librería) en lenguaje Smalltalk que permite representar agentes, recursos y ambiente. La especificación del comportamiento de estas entidades (agente, recurso, ambiente) debe programarse usando el lenguaje Smalltalk, lo cual implica poseer conocimientos del mismo, siendo una de las dificultades en el uso del framework.
Esto motivó la búsqueda de otro framework que presentara similares prestaciones que Cormas pero que además estuviera desarrollado en Java. Por lo tanto, se estableció Cormas como la base para poder definir criterios de evaluación para la extensión o creación de un nuevo framework.

En este capítulo se detallan los criterios de evaluación establecidos, junto con un conjunto de frameworks evaluados y los resultados obtenidos.

3.1. Criterios de evaluación

El framework que buscamos debería cumplir como mínimo con los criterios que se marcan como imprescindibles. Al momento de decidirnos por un framework en particular, se priorizan aquellos que además cumplen con algunos de los criterios deseables.
Los criterios de evaluación son los siguientes:

3.1.1. Criterios Imprescindibles

Documentación - Es imprescindible que el framework cuente con la mayor documentación posible. Basándonos en el tipo de documentación disponible es que se definen los siguientes valores:

Ninguna – No se tuvo acceso a la documentación.
Escasa – Sólo documentación incluida en la distribución del framework, y/o ejemplos de uso.
Básica – *Escasa* + Manuales de usuario, guías de instalación, artículos recomendados, FAQ.
Completa – *Básica* + manuales técnicos o tutoriales, respaldo de una comunidad, contacto con desarrolladores, newsgroups, ejemplos de código, API.

Lenguaje de Programación - El lenguaje de desarrollo del framework debe ser Java. Esto es un requisito establecido en el presente proyecto, ya que en caso de ser necesario se quiere poder desarrollar código Java para su extensión. Excepcionalmente consideraremos el framework .NET (y el desarrollo en alguno de los lenguajes soportados por este) si es que se encuentra algún framework que se destaque del resto, en cuanto a los demás criterios aquí descritos.
Además del lenguaje de desarrollo del framework, es necesario conocer el lenguaje en que se especifican (mediante codificación) los modelos. Por tal motivo nos centraremos en el lenguaje de desarrollo y en el lenguaje para la especificación de modelos que provee el framework.

Licencia - Se investigará el tipo de licencia provista. El software debe encontrarse disponible de forma gratuita. Sin embargo, existen diferencias en cuanto a la disponibilidad del código fuente en programas de libre acceso debido a sus licencias.
Por ejemplo en el caso de software de código fuente abierto, liberado bajo GPL o LGPL el código es visible, puede ser adaptado o incluso ampliarse. Por lo tanto, nos enfocaremos en aquellos frameworks que cuenten con licencias del tipo GPL o LGPL ya que facilitan la tarea de extensión del código fuente.

Entorno de simulación - Es necesario que el framework provea un entorno de simulación. Dicho entorno tendrá que admitir la simulación de modelos basados en SMA.

Mecanismos de extensión - El framework deberá permitir aplicar mecanismos de extensión. Se detallará cual es el utilizado en caso de contener uno. Además, se investigará la flexibilidad que posee para interoperar con otros plugins y/o librerías.

3.1.2. Criterios Deseables

Modelado – Es deseable que el framework provea mecanismos para representar las siguientes entidades: ambiente, recursos y agentes.
En particular se investigará si es que el framework permite incorporar aspectos específicos de un dominio, ya que nuestro estudio se basa en simulaciones de agro-ecosistemas.

Se investigará la forma por la cual se especifica el comportamiento y la comunicación entre dichas entidades.

Actualización y Mantenimiento - Es deseable que el framework este actualizado, tenga mantenimiento y que no sea una herramienta en desuso. Se tendrá en cuenta aquellos que han lanzado versiones estables en estos últimos años. Verificaremos si las funcionalidades provistas están incompletas y si es así, si existe un detalle de ello o si se está trabajando para ser incluidas en una posterior liberación. También se observará la existencia de reportes sobre bugs. Para poder realizar una distinción entre los

frameworks especificamos la última fecha en que se realizó una liberación y definimos los siguientes estados:

> *Mantenido* – Que se encuentren actualizaciones del framework, que cuente con un desarrollo continuo y liberaciones de versiones recientes.
> *En desuso*- Que no cuente con mantenimiento, que la última versión halla sido liberada hace varios años y/o que no se conozca continuidad en su desarrollo.
> *Incompleto*- Que la versión liberada no esté completamente desarrollada o sea inestable y/o se desconozca la continuidad del framework.

3.2. Frameworks Evaluados

Se investigó una lista de veintisiete frameworks. Se puede ver el detalle de los mismos en el Apéndice B.
Al momento de evaluar los frameworks, es necesario destacar, que la curva de aprendizaje para adquirir un conocimiento práctico de los mismos es considerable. Para que sean válidas las declaraciones sobre algunos criterios, se debería realizar un exhaustivo estudio de cada uno de ellos, lo cual queda por fuera del alcance del presente proyecto. Por esta razón, la primera evaluación y selección de frameworks está basada en la información secundaria analizada. Las razones de exclusión se fundamentan según la documentación técnica investigada.

En la Tabla 3.1 se marcan con cruces aquellas características que no son cumplidas por los frameworks o no se pudieron determinar y por tanto serán consideradas razones de exclusión.

Framework	Documentación no completa (básica o escasa).	Lenguaje no Java (desarrollo o especificación).	Licencia no libre, código fuente no disponible o versiones disponibles son trial.	No posee un entorno de Simulación propio, o el entorno de simulación no cuenta con las características deseadas.	Herramienta en desuso, poco mantenida o incompleta.	No posee un mecanismo de extensión.
AgentSheets			X			X
AgentTool	X			X		
Breve	X	X				
Cormas	X	X				
ECHO	X				X	X
Ingenias				X		
Jack		X	X			
JADE				X		
Madkit	X			X	X	X
MAGSY	X	X	X			X
MAML	X	X		X	X	
MIMOSE	X	X	X			X
NetLogo		X	X			
PS-i	X	X	X		X	X
SimAgent	X	X			X	
SimPack		X				X
StarLogo		X	X			X
SugarScape	X				X	
Swarm	X	X			X	
TeamBots	X			X	X	X
VSEit	X		X			
Zeus	X					X

Tabla 3.1: Frameworks no seleccionados y razones de exclusión.

Los frameworks seleccionados son aquellos que cumplían satisfactoriamente con los criterios marcados como imprescindibles y que contenían además características deseables. Esto permitió acotar la lista inicial a cinco frameworks que se detallan a continuación:

- MASON ("**M**ulti-**A**gent **S**imulator **O**f **N**eighborhoods")
- Mimosa
- RePAST (**Re**cursive **P**orus **A**gent **S**imulation **T**oolkit)
- SeSAm (**Sh**ell for **S**imulated **A**gent Syste**m**s)
- Ascape

El framework Mimosa, si bien no cumple con algunos de los ítems (cuenta con una documentación básica y la versión liberada es beta), no se excluyó por considerarla una herramienta "sucesora" de Cormas, ya que su impulsor (Jean-Pierre Muller) pertenece al mismo grupo que desarrolló Cormas (grupo GREEN del CIRAD [22]).
Mimosa brinda similares prestaciones y básicamente tiene el mismo entorno de simulación que Cormas, pero se destaca por su programación en Java.
Al momento de investigar los frameworks seleccionados, el análisis de Mimosa se realizará desde otra óptica, evaluando si en el presente proyecto se podría realizar una posible extensión o continuidad.

Luego de la realización de un primer análisis, se investigó a un nivel más detallado los cinco frameworks anteriormente mencionados.
Se realizó la instalación de los frameworks, se desarrollaron ejemplos prácticos y se profundizó la investigación teniéndose en cuenta los criterios que se detallan en la siguiente sección.

3.3. Criterios de Evaluación para el estudio extendido de los frameworks seleccionados

A continuación se especifican un conjunto de criterios los cuales fueron guía para la realización de un estudio en profundidad de los frameworks MASON [23], Mimosa [24], RePAST [25], SeSAm [26] y Ascape [27].

3.3.1. Criterios Generales

Soporte – Se investigará si existen *listas de mails*, si posee el respaldo de una *comunidad* y si existe algún *contacto con los desarrolladores*.

Integración con Eclipse – Si bien los frameworks elegidos están desarrollados en Java, es deseable detallar si poseen un *IDE (*Integrated Development Environment*) propio* o pueden integrarse con facilidad a Eclipse como *plugin* o como una *librería*.

Uso – Analizaremos la simplicidad en el uso, si es una herramienta intuitiva, si requiere conocimientos de programación, detallando el nivel de dificultad. Según los conocimientos de programación necesarios para su uso, definimos tres niveles:

> *Ningún conocimiento de programación* – No es necesario el desarrollo de código por parte de un usuario y no es necesario contar con nociones de estructuras de programas, el usuario no necesita ser un programador.

Conocimientos básicos - Es necesario que el usuario tenga conocimiento sobre estructuras de programas y deba realizar codificación en algún lenguaje de programación.
Conocimientos medios o altos – El usuario necesita contar con sólidos conocimientos en programación, o sea, para hacer uso del framework necesita ser programador.

3.3.2. Criterio respecto el diseño del framework

Organización del código y estructura interna – Si el diseño está basado en algún estilo de arquitectura y si presenta una estructura compuesta por *módulos*, por *librerías* o agrupada en *paquetes*. Se especificará el estado en que se encuentra el código fuente (*comentado*, *organizado*, *confuso)*.

3.3.3. Criterios respecto del modelado

Especificación del modelo – Investigaremos si es posible especificar en forma gráfica (por medio de un *entorno gráfico*) los modelos o es necesario el *desarrollo de código* (y en tal caso si el código es Java). Esto se analizará tanto para la especificación de diagramas de estructura como diagramas de comportamiento (si permite realizar diagramas de clases, diagramas de actividad, u otro). Se evaluará si este modelado cumple con algún estándar (por ej. UML).
Se investigará cómo se definen los modelos, ya sea definiendo ontologías, entidades u otros. Además analizaremos si se permite representar *recursos, agentes* y/o *ambiente*.

Formato de almacenamiento – Para el caso en que el framework permita persistir el modelo y/o los resultados de la ejecución de una simulación, es deseable determinar el formato de almacenamiento (por ej.: XML).

3.3.4. Criterios respecto de la simulación

Definición de una simulación – Investigaremos si es posible especificar en forma gráfica la simulación (por medio de un *entorno gráfico*) o es necesario el *desarrollo de código* (y en tal caso si el código es Java). Se verificará si se puede realizar la inicialización y definición de variables por medio gráfico o código y el grado de dificultad del desarrollo de la simulación (*simple*, *medio*, *compleja*).
Se analizarán algunas cualidades de visualización gráfica cuando corresponda (definición de grillas, definición de vistas para las celdas u otra.).

Manipulación de datos de salida de la simulación – Se investigará si los datos pueden ser exportados (y si es así en que formato) y/o si a partir de ellos se pueden realizar *gráficos* y/o *estadísticas.*

Avance del tiempo – Se define la forma de avance del tiempo, como *time-step*, *tiempo continuo* o *tiempo discreto*.

Tipo de inicialización – Se investigará el tipo de inicialización provisto, o sea, si pueden generarse *valores personalizados* y/o *aleatorios.*

Uso de una semilla – Verificaremos *Si* permite generar números aleatorios a partir del uso de una semilla, ya que si partimos siempre de la misma semilla podemos obtener

la misma secuencia de números aleatorios. Esta es una característica interesante para reproducir exactamente una simulación pasada, o sea, los mismos resultados de una simulación partiendo de una misma condición inicial.

Controles de ejecución - Detallaremos cuales con los controles provistos, por ejemplo: ejecución step by step o ejecución continua de steps.

Acceso a la simulación – Investigaremos las posibilidades de acceso a una simulación, para ello definimos las siguientes formas:

- *Almacenamiento de inicializaciones* – si se permite volver a realizar una simulación con los mismos valores con la cual inicializamos una simulación previa.
- *Re anudamiento* – si se permite reanudar una simulación desde algún punto de ejecución.
- *Grabación* – si se permite grabar una simulación y volver a ejecutarla.
- *Logs* – si mantiene algún archivo de logs que registre la ejecución de la simulación.

Otras características

Se detallarán aquellas características particulares provistas por los frameworks, que sean consideradas de utilidad. Estas funcionalidades serán consideradas como un aporte adicional a la hora de comparar frameworks similares.

3.4. Resultados del estudio extendido para los frameworks seleccionados

En la Tabla 3.2 se resumen las características de cada framework investigado, basado en los criterios definidos en las secciones 3.1 y 3.3.
Para aquellos ítems para los cuales no se encontró información, se marcan con el término "*No disponible*" y para aquellos en que no cumple o verifica con alguno de los parámetros provistos, se marcan con el término "*No verifica*".

Características		MASON	Mimosa	RePAST	SeSAm	Ascape
Criterios Generales						
Actualización y Mantenimiento	**Estado**	Mantenido	En desuso, Incompleto.	Mantenido	Mantenido	Mantenido
	Última liberación	Año 2008	19/04/2006	Año 2009	19/01/2009	20/04/2009
Documentación		Completa	Básica	Completa	Completa	Completa
Lenguaje de desarrollo/especificación		Java/Java	Java/Java script, Jess, Python, Prolog, Smalltalk	Java/Java	Java/ No disponible	Java/ No disponible
Licencia		AFL v3.0	LGPL	BSD	LGPL	BSD
Mecanismos de extensión		Si, codificando	Si, codificando	Si, codificando	Si, codificando	Si, codificando
Soporte		Lista de Mails	Lista de Mails	Lista de Mails	Lista de Mails, Comunidad, Contacto con desarrollado-res	Lista de Mails, Contacto con desarrollado-res
Integración con Eclipse		Librería	IDE Propio	Plugin	IDE Propio, Librería	Librería, Plugin
Uso (según niveles de conocimiento en programación)		Conocimiento medio o alto	Conocimiento básico	Conocimiento medio o alto	Ningún conocimiento	Conocimiento medio o alto
Diseño del Framework						
Organización del código y estructura interna	**Estado del código**	Comentado, Organizado.	Comentado	Comentado	Confuso	Comentado, Organizado.
	Estructura del framework	Paquetes	Paquetes	Modular, Paquetes	Paquetes	Modular, Paquetes
Modelado						
Especificación del modelo	**Medio**	Des. de código.	Entorno Gráfico,Des. de código.	Entorno Gráfico, Des. de código.	Entorno Gráfico	Des. de código.
	Representación de entidades	No verifica	No verifica	Agente, Ambiente	Agentes, Recursos, Ambiente	Agentes, Recursos
Formato de almacenamiento	**Modelo**	.java	.xml	.java, .agent, .groovy	.xml	.java
	Resultados	.checkpoint	.xml	.xml	.xml	No disponible
Entorno de Simulación						
Definición de una simulación	**Medio**	Desarrollo de código.	Entorno Gráfico	Entorno Gráfico, Desarrollo de código.	Entorno Gráfico	Desarrollo de código.
	Grado de dificultad	Media	No disponible	Media (des. código) Compleja (ent. grafico)	Compleja	Media
Manipulación de datos de salida		Gráficos	Snapshot	Snapshot	Gráficos, Snapshot	Gráficos, Estadísticas
Avance del tiempo		Time-Step	Time-Step	Time-Step	Time-Step	Time-Step
Tipo de inicialización		Valores pers.	Valores pers.	Valores pers.	Valores pers. y aleatorios.	Valores pers.
Uso de semilla		Si	Si	Si	No disponible	No disponible
Controles de ejecución		Ejecutar, Terminar ejecución, Pausar.	Inicializar, Ejecutar, Ejecutar paso a paso, Terminar ejecución, Reiniciar.	Inicializar, Ejecutar, Ejecutar paso a paso, Reiniciar.	Ejecutar, Pausar, Reiniciar, Ejecutar paso a paso.	Inicializar, Reiniciar, Terminar ejecución, Pausar, Ejecutar paso a paso.
Acceso a la simulación		Reanuda-miento, Grabación	Logs	Grabación, Logs	Almacena-miento de inicializacio-nes, Logs	Reanuda-miento, Grabación

Tabla 3.2: Resumen de características de frameworks seleccionados.

Luego del estudio extendido y en base a la implementación de un ejemplo práctico basado en el caso de estudio (ver capitulo 5) se llegó a las siguientes conclusiones:

Todos los frameworks considerados cuentan con un entorno de simulación similar, en lo que refiere a funcionalidades y capacidades, por tanto nos enfocaremos en el modo de especificación y representación de los modelos y no tanto en el entorno de simulación.

Mimosa, no cuenta con representaciones de las entidades "claves" en lo que se refiere a SMA (agente, recurso y ambiente). Además, al encontrarse en una versión beta, no ser una herramienta estable (comprobando su inestabilidad mediante la implementación del ejemplo) y no tener mantenimiento, resulta inviable considerarla; ya que inicialmente se debería continuar con su desarrollo (lo cual queda por fuera del alcance de este proyecto), para luego extenderla. Esto requeriría un esfuerzo y dedicación adicional, para obtener un framework que cumpla con los requisitos estipulados, encontrándose otras herramientas con mayor madurez en este estudio.

MASON, un aspecto negativo de este framework al igual que Mimosa es que no cuenta con representaciones de las entidades agente, recurso y ambiente. Es decir, no provee clases específicas que representen a las distintas entidades que conforman el dominio que queremos modelar.

RePAST, no posee clases específicas para el modelado de las entidades agente y recurso. Además, el entorno gráfico que provee para la definición de modelos resulta poco intuitivo, ya que su representación no cumple con ningún estándar de modelado (por ej. UML), dificultando la especificación de estructura y comportamiento mediante los diagramas provistos para ello.

SeSAm, posee una representación para todas las entidades involucradas en nuestra realidad (agente, recurso, ambiente).
Dado que sólo provee un entorno gráfico (IDE) para la especificación del modelado, se investigó el código fuente (provisto por la distribución) para obtener más detalles del framework y así poder determinar el grado de interoperabilidad. El código fuente no cuenta con el conjunto de librerías necesarias para su correcto funcionamiento, ni tampoco una especificación de éstas, lo cual imposibilita la compilación y ejecución desde Eclipse. Además, no se provee de ningún tipo de documentación que especifique la estructura de paquetes ni el kernel, dificultando la comprensión y entendimiento del código, siendo además una aplicación con más de 3000 clases.

Ascape, posee una representación para las entidades agente y recurso. Si bien no provee explícitamente una representación para la entidad ambiente, esta podría implementarse fácilmente.
Este Framework no provee un entorno gráfico para definir el modelo, debiéndose desarrollar desde un IDE (por ej. Eclipse). No se encuentra documentación que detalle la estructura de paquetes ni del kernel. Sin embargo, el código se encuentra bien comentado, lo cual resulta útil para determinar las responsabilidades de las clases. La distribución que provee el código fuente contiene las librerías necesarias para poder compilar y ejecutar la aplicación la cual cuenta con aproximadamente 700 clases.

En base a lo antes expuesto, los frameworks que mejor se adecúan a los criterios estipulados, son Ascape y SeSAm. Ambos pueden ser extendidos y proveen los

conceptos necesarios para el modelado y simulación de SMA. Sin embargo, considerando la estructura, tamaño, especificación, documentación y dificultad dada por la disponibilidad del código fuente provista por SeSAM, concluimos que Ascape es el framework que mejor se adapta a nuestras necesidades, ya que será necesaria su extensión.

Capítulo 4

Construcción de FW4SimSMA

En este capítulo se describen las etapas que determinaron la construcción de FW4SimSMA. FW4SimSMA se ejecutará sobre la plataforma Eclipse, esto es para aprovechar al máximo las ventajas que da el uso de plugins dentro del IDE. También, detallaremos brevemente los frameworks utilizados (EMF [28], GMF [29] y Acceleo [30]) y el rol que éstos tuvieron en el desarrollo de FW4SimSMA.

4.1. Etapas para la construcción de FW4SimSMA

La construccion de FW4SimSMA está compuesta de las siguientes etapas (ver Figura 4.1): definición del modelo basado en EMF, desarrollo de un editor gráfico realizado con GMF, construcción de la simulación (generación de código) basada en Acceleo y ejecución de la simulación basada en el framework seleccionado (Ascape).
Como objetivo, FW4SimSMA debía integrarse a la plataforma Eclipse. Esto es para aprovechar al máximo las ventajas que da el uso de plugins dentro del ambiente de Eclipse. Para ello, se escogieron EMF y GMF para el desarrollo de nuestro modelo y el editor gráfico respectivamente. GMF permite la representación gráfica de diagramas basados en modelos definidos en EMF. Estas etapas son estándar cuando se hace uso de éstos frameworks. Luego, una vez finalizadas las etapas anteriores, mediante el uso de Acceleo se realiza la generación semiautomática de código Java (necesario para la ejecución de una simulación) para los diagramas especificados con el editor gráfico y el modelo EMF. Acceleo permite una completa integración a Eclipse y a EMF, lo cual determinó su elección. Como última etapa, el código antes generado es enriquecido y utilizado para la ejecución de una simulación a través del framework Ascape.
A continuación se detallan las etapas antes descriptas:

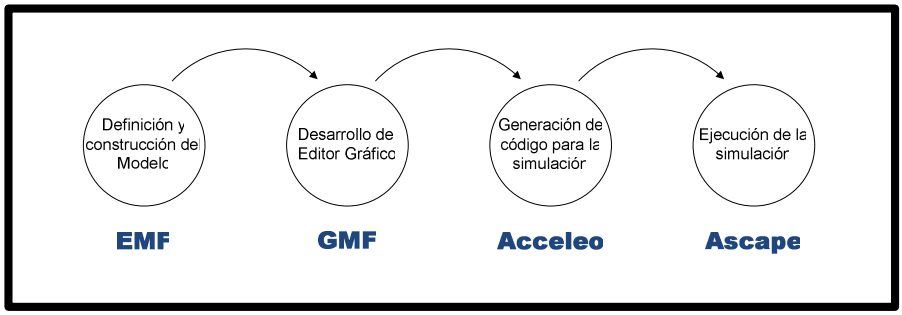

Figura 4.1: Etapas para la construcción de FW4SimSMA.

4.2. Definición del modelo basado en EMF

Esta etapa consiste en dos pasos: el primero es la definición del modelo (metamodelo), a partir del cual se define la estructura, semántica y restricciones para la construcción de modelos en los cuales se podrán representar los conceptos significativos de nuestro dominio y el segundo paso es la construcción del modelo antes definido, utilizando Eclipse Modeling Framework (EMF).
EMF es un framework para el entorno Eclipse que provee facilidades para la definición de modelos ya sea como código, como un diagrama UML o como una descripción en XML, además de permitir realizar transformaciones entre éstas distintas representaciones.
A continuación se define el modelo y se detalla su construcción en EMF:

4.2.1. Definición del modelo

En el modelado se definen distintos tipos de entidades, algunas de ellas para la especificación de diagramas de estructura estática (particularmente diagramas de clase) las cuales se basan en AML y otras para la especificación de diagramas de comportamiento (particularmente diagramas de actividad) basadas en UML. AML se basa en UML, pero incorpora varios conceptos de modelado los cuales son apropiados para capturar las características típicas de los SMA, por tal motivo utilizamos AML como base para nuestros modelos.

Para la especificación de diagramas de estructura, se definió la entidad *Diagrama Clase*, la cual es contenedora de las entidades *Agente*, *Recurso*, *Ambiente*, *Agentes_Localizados*, *Recursos_Localizados* y *Celda*. Las entidades *Agentes_Localizados* y *Recursos_Localizados* representan a los agentes y recursos que se alojan en un lugar determinado del ambiente (*Celda*), permitiendo que a lo largo de la simulación los mismos puedan re-ubicarse en una nueva celda. Para las entidades antes mencionadas se pueden definir asociaciones y atributos.

Para la especificación de diagramas de actividad, se definió la entidad *Diagrama_Actividad*, la cual es contenedora de las entidades *Condición*, *Metodo*, *Actividad*, *Ciclo*, un elemento de *Inicio* de flujo y un elemento de *Fin* de flujo.
Una *Condicion* representa a una toma de decisión con dos posibles salidas. Para ello se evalúa una condición booleana. Un *Método* representa la invocación a un método el cual determina los envíos de mensajes entre los elementos de la realidad. Una *Actividad* sólo tiene un nombre y determinará un destino al cual se debe continuar una vez que ésta terminó. Un *Ciclo* representa la ejecución repetida de un flujo (*Diagrama_Actividad*) y las entidades *Inicio* y *Fin* marcarán el inicio y fin de un flujo, respectivamente. Las entidades pueden tener o no conectores y de tenerlos puede variar en uno o más conectores que determinan el orden del flujo. La cantidad de conectores dependerá de cada entidad según la definición del mismo, es decir, un elemento *Condición* conduce a uno o dos posibles entidades, uno cuando la condición se evalúa en true y la otra en false. Un *Método*, *Actividad*, *Ciclo* y elemento de *Inicio* conducen a solo una entidad en el flujo y la entidad *Fin* no conduce a ningún elemento en el flujo.
Finalmente, dado que se quiere vincular a ambos diagramas es que se asocia una entidad *Clase* con la entidad *Diagrama_Actividad*. Mediante esta relación representamos que cualquier clase puede tener definido un comportamiento el cual se ejecutará en cada paso de la simulación.

La Figura 4.2 presenta el modelo que representa la realidad de nuestro problema. Las restricciones definidas para el modelo pueden verse en detalle en el Apéndice C.

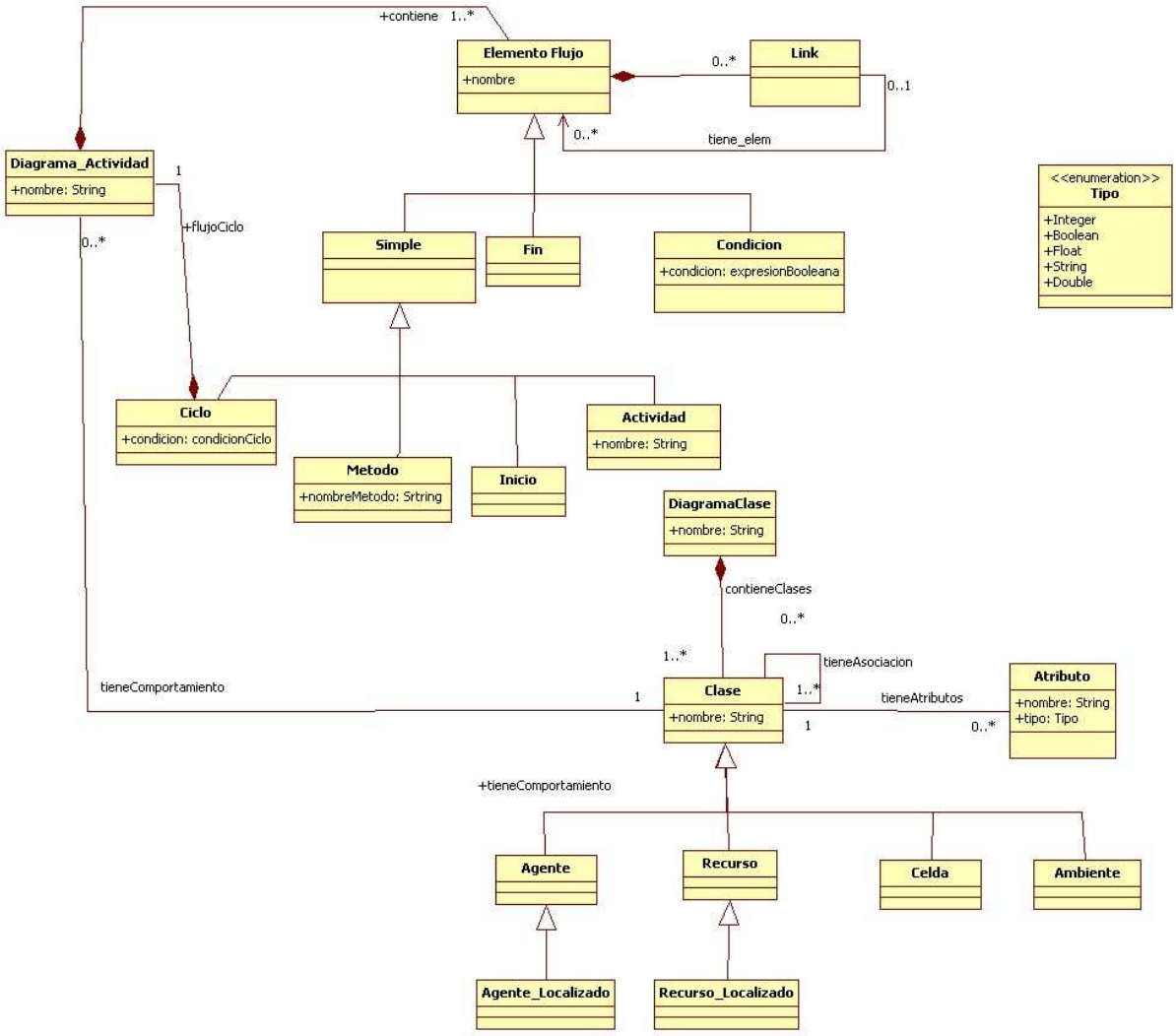

Figura 4.2: Modelo.

4.2.2. Construcción del modelo en EMF

El modelo utilizado para representar modelos en EMF es llamado ecore. El modelo ecore puede ser creado utilizando directamente una serie de editores de EMF, a través de herramientas gráficas o convirtiendo modelos UML o expresados en formato XMI [31].

Para definir el modelo, se comienza creando un diagrama ecore, al mismo tiempo que EMF genera un documento XML con extensión .ecore, que asocia al diagrama anterior y el cual será utilizado para la generación de código.
En este proyecto, se requiere poder crear diagramas de clase y diagramas de actividad, pero es necesario relacionarlos, dado que el diagrama de actividad define el comportamiento específico (que se ejecutará en cada step) que tendrá una *Clase* definida en el diagrama de estructura. Por lo tanto se desarrolló un solo modelo (un

único archivo ecore_diagram) al cual se le asocian dos archivos ecore. Esto permite definir propiedades con distinto valor según corresponda para uno u otro diagrama, usando una misma definición de modelo.

4.3. Desarrollo del editor gráfico con GMF

Graphical Modeling Framework (GMF) está basado en EMF y en GEF (Graphical Eclipse Framework). GMF permite a partir de modelos, generar automáticamente editores gráficos como plugins para Eclipse.

La construcción de un editor grafico con GMF está compuesta por varios pasos que se detallan en la Figura 4.3.

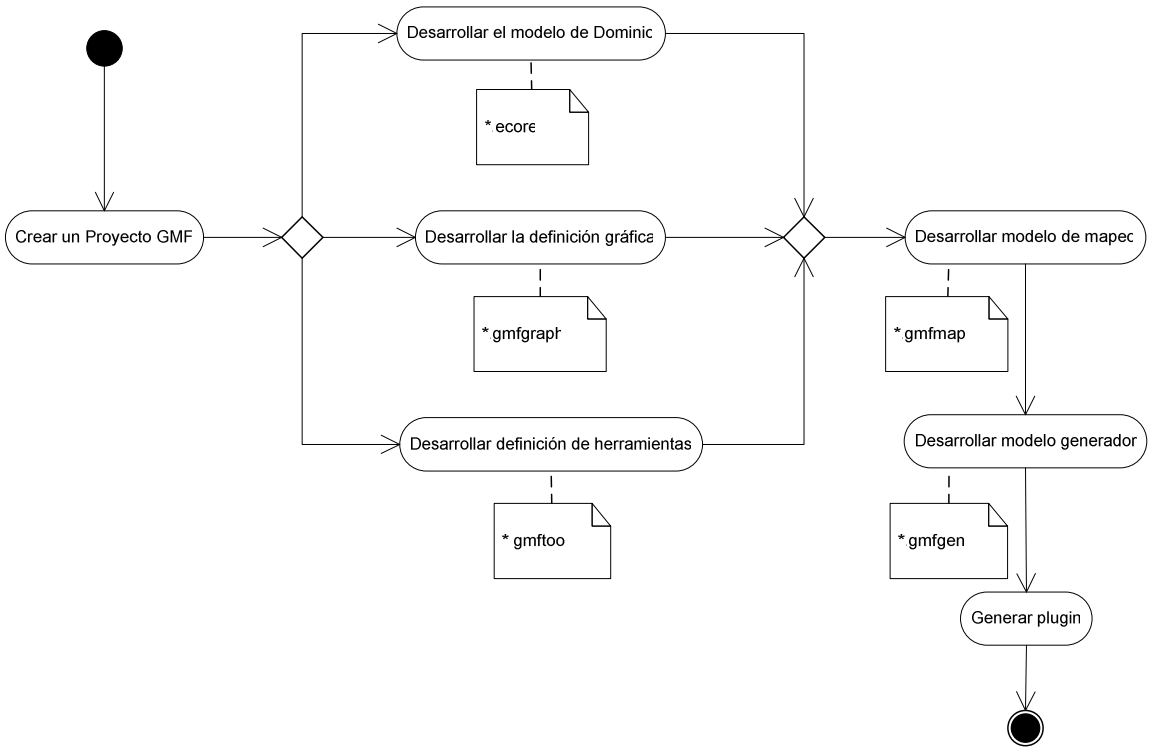

Figura 4.3: Pasos para la construcción de un editor grafico con GMF

A continuación se describen los pasos que componen el desarrollo del editor gráfico.

4.3.1. Creación de un proyecto GMF

Para la creación de un proyecto GMF se debe instalar el plugin GMF en la plataforma Eclipse, que proveerá un framework en donde crearemos nuestro proyecto GMF.

4.3.2. Desarrollo del modelo

El objetivo principal de GMF es proporcionar un editor gráfico para modelar diferentes entornos partiendo de un modelo o metamodelo de entrada. El modelo se puede crear dentro de GMF desde cero o importar un modelo realizado previamente. La segunda opción fue la realizada. Inicialmente se definió el modelo con EMF y luego se realizó la importación a GMF. El modelo importado es el definido en la sección 4.2.1.

4.3.3. Importación del modelo a GMF

Tanto en la construcción del diagrama de clases como en la del diagrama de actividad se importaron los modelos realizados previamente. Esto provocó la generación de dos proyectos GMF: uno para la edición de diagramas de clase y otro para diagramas de actividad. Por tanto, en cada proyecto GMF luego de la importación correspondiente se generó un archivo de extensión .genmodel, que se relaciona con el modelo importado de extensión .ecore_diagram y .ecore respectivamente.

4.3.4. Desarrollo de la definición gráfica

La definición gráfica del modelo consiste en definir el aspecto que van a tener las primitivas de modelado en el editor gráfico de forma semiautomática. GMF permite personalizar el aspecto de la aplicación de construcción de modelos específicos de dominio según las necesidades del desarrollador.
La definición gráfica (con extensión .gmfgraph) define los elementos visuales utilizados para representar los elementos del dominio en el editor gráfico. Los elementos pueden ser definidos con base en la composición de figuras geométricas básicas (triángulo, rectángulo, elipse, línea), definiendo la posición, tamaño, color, entre otras propiedades de cada figura, que será utilizada para representar a los elementos del modelo.
Este paso no resulta ser tan automático como lo indican los tutoriales del framework, pues es necesario definir manualmente muchas de las propiedades que se detallan a continuación.

Particularidades en el desarrollo

Para la definición gráfica de los diagramas de clase, como era deseable que la visualización de entidades se basara en el estándar AML y UML, fue necesario realizar figuras compuestas, ya que la generación semiautomática no generaba la representación deseada.
A continuación, en la Figura 4.4 se ilustra la representación gráfica de la entidad *DiagramaClase*, la cual constó de la composición de dos figuras rectángulo, una en donde se ingresa el nombre de la clase y la otra en donde se definirán los diferentes elementos del diagrama de clase.

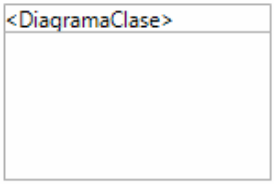

Figura 4.4: Representación gráfica de la entidad *DiagramaClase*

La representación gráfica para *Agente*, *Recurso*, *Agente_Localizado*, *Recurso_Localizado*, *Celda* y *Ambiente* se ejemplifica en la Figura 4.5, en las cuales dentro de una figura rectángulo se dibujan otros dos, que permiten ingresar el nombre de la clase y atributos respectivamente. También puede observarse que en el extremo superior izquierdo de cada representación se encuentra una imagen basada en AML que define a cada entidad del SMA.

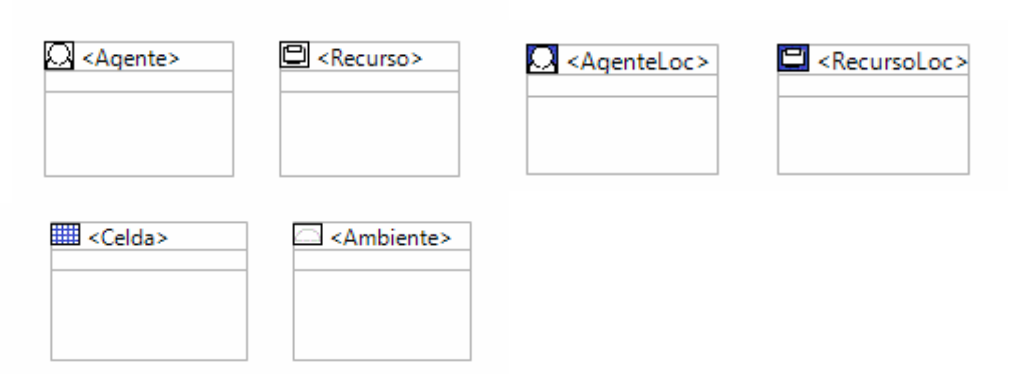

Figura 4.5: Representación gráfica de entidades que contienen atributos.

Por último, en la Figura 4.6 se ilustra la representación gráfica que se desarrolló para los atributos y sus tipos (por ej.: Integer, el cual es el tipo de datos establecido por defecto).

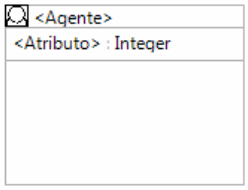

Figura 4.6: Representación gráfica de atributos.

4.3.5. Desarrollo de la paleta de herramientas

La paleta de herramientas (con extensión .gmftool) representa los elementos visuales de dibujo que tendrá el editor, tales como menús o botones. Por cada elemento que se quiere representar en la paleta de herramientas, se debe crear un objeto, al cual por defecto se le crean dos imágenes que pueden referenciarse con las imágenes deseadas (ser personalizadas). Una imagen corresponde a como se visualiza en la paleta y la otra a la imagen que aparece en la esquina superior izquierda de la entidad cuando es dibujada.
En nuestro caso se redireccionaron a las imágenes deseadas basadas en el estándar de UML y AML, tanto para el diagrama de clases como también para el de actividad.

4.3.6. Desarrollo del mapeo

En el proceso de mapping se unen todos los pasos previos que permiten formar la herramienta de construcción de modelos específicos de dominio. En este paso es donde se determina si toda la creación de la herramienta tiene consistencia. En este punto del proceso se tienen tres modelos: el modelo, de definición gráfica y la definición de la paleta de herramientas. Para unir todos los modelos y definir qué elementos corresponden a qué elementos entre los distintos modelos, es necesario crear un mapeo (con extensión .gmfmap) que una todos los conceptos representados en los tres anteriores. En el desarrollo del mapeo se utilizan los elementos del modelo, asignándoles una figura del modelo gráfico y el elemento del modelo de la definición gráfica de la paleta con la que se dibujarán.
Al igual que en el desarrollo de la definición gráfica, este paso no resulta ser tan automático. Por tal motivo es necesario definir manualmente muchas de las propiedades que se detallan a continuación.

Particularidades en el desarrollo

Dado que los modelos EMF no logran una especificación precisa debido que la notación gráfica no permite representar completamente un modelo, se utiliza Object Constraint Lenguaje (OCL) [33] para describir restricciones adicionales sobre los objetos del modelo. Muchas veces estas restricciones se describen en lenguaje natural con el riesgo de ocasionar ambigüedades. Para escribir especificaciones correctas se han desarrollado los lenguajes formales. OCL es un lenguaje formal para la descripción textual y precisa de restricciones que se aplican a los modelos UML.
En este paso es donde se definen las restricciones OCL que permiten validar el modelo. Las mismas pueden definirse sobre alguna asociación creada o sobre la propia entidad. En nuestro caso particular se realizó en ambas. GMF provee para la definición de estas reglas una especificación híbrida que toma a OCL como base, con lo cual la definición de reglas sobre los modelos no resultó una tarea simple de realizar, ya que además no provee de un validador sintáctico para la verificación de las mismas.

4.3.7. Desarrollo del generador de modelos

Una vez finalizado el mapeo, ya está el proceso de definición de la herramienta completa, solo está faltando generar su código (con extensión gmfgen). Este modelo es generado por GMF a partir del modelo de mapeo y sólo se modifica para personalizar las opciones de generación del editor visual.

Fue necesario personalizar algunas de las propiedades que permiten validar las expresiones OCL definidas para el modelo.

Esto fue todo lo que se requirió para crear el editor visual en GMF. Una vez generado el código solo se necesita ejecutar la aplicación como un plugin de Eclipse y verificar su funcionamiento.

4.4. Desarrollo del módulo de generación de código

El modulo de generación de código desarrollado utiliza el framework Acceleo [30], el cual permite de forma sencilla a partir del modelo definido con EMF y los modelos definidos por el usuario mediante el editor grafico generado con GMF, la generación de código Java enriquecido con las estructuras necesarias para su ejecución mediante el framework Ascape.

Algunas de las principales características de Acceleo son:

- Generación de código basada en plantillas que expresan reglas de transformación entre un modelo y el código, utilizando una sintaxis similar a Java Server Pages (JSP).
- Una completa integración con el ambiente de Eclipse y el framework de EMF.
- Navegación por los elementos de cualquier modelo que siga los estándares de EMF (XMI).
- Simplicidad en el mantenimiento y actualización de todas las plantillas.
- Coloreado sintáctico de las plantillas, así como detección de errores basada en un metamodelo.

Acceleo proporciona un editor de templates el cual ofrece un conjunto de instrucciones que permiten realizar ciclos, tomar decisiones y navegar por los elementos del modelo de una manera muy natural. Esto provee una forma de integrar conceptos relativos a una plataforma específica dentro de los templates y extraer esos conceptos de los modelos de manera que éstos puedan ser independientes de la plataforma.
A continuación se definen los templates necesarios para la generación de código de nuestro dominio.

4.4.1. Definición de templates

Para cada una de las construcciones definidas en el editor de diagramas de estructura estática (diagrama de clases), se definen los templates necesarios a partir de los cuales por medio de Acceleo se genera el código Java correspondiente.

En la Tabla 4.1 se especifican los templates desarrollados y sobre que entidades definen las reglas de transformación para la generación de las clases Java correspondientes.

Template	Entidad
Agent.mt	Agente
Cell.mt	Celda
Environment.mt	Ambiente
LocatedAgent.mt	Agente_Localizado
LocatedResource.mt	Recurso_Localizado
Resource.mt	Recurso

Figura 4.1: Templates definidos

El template **Model.mt** difiere de los anteriores en que no mapea una entidad especifica, este es responsable de generar el código de la clase Model, la cual es la clase encargada de definir la relación entre las diferentes clases definidas así como las estructuras necesarias para la ejecución de la simulación en el framework Ascape.

El detalle completo de la estructura de estos templates y su funcionamiento puede consultarse en el Apéndice D.

4.4.2. Invocación a Acceleo

La invocación a Acceleo para realizar la generación de código involucra el pasaje del modelo explicado en la sección 4.2.1, los siete templates necesarios para el parseo y el modelo específico definido por el usuario.

Dado que el proceso de modelado utilizando FW4SimSMA, tiene como resultado la generación de un diagrama de estructura estática y varios diagramas de comportamiento y considerando además, que Acceleo toma un único XML del cual se genera el código, es necesario realizar una unión de los diagramas mencionados generando un archivo temporal el cual será consumido por Acceleo.

Para lograr identificar los diagramas de comportamiento se definió una nomenclatura de forma de facilitar la unión de los archivos.

Los nombres de los diagramas de comportamiento definidos tienen la siguiente estructura:
nombre-dinamico_entidad_nombre-entidad.agroecoMAD, donde **nombre-dinamico** refiere al nombre del diagrama de estructura estática, **entidad** refiere a la entidad a la que corresponde el comportamiento (agente, agente_localizado, recurso, recurso_localizado) y **nombre_ entidad** corresponde al nombre de la entidad.

4.4.3. Integración de los módulos

Para lograr un único framework con los módulos de generación de los diagramas antes definidos y la generación de código, se utilizaron la extensiones para plugin ofrecidas por Eclipse, la cuales permiten agregar de forma sencilla nuevas funcionalidades a dicho IDE.

En la Figura 4.7 se ilustran las extensiones realizadas. A continuación se menciona la acción de cada una.

- **agroecosistemas.mas.code.gen:** Mediante esta extensión se habilita la opción de generación de código al realizar click derecho sobre un archivo de extensión .agroecoMA. La acción que se ejecuta al seleccionar dicha opción invoca el modulo de generación de código.
- **agroecosistemas.mas.code.gen.definirComportamiento.Agente:** Mediante esta extensión se habilita la opción de definición de comportamiento al realizar click derecho sobre una entidad *Agente* definida en el diagrama de estructura estática. La acción que se ejecuta al seleccionar dicha opción invoca al editor de diagramas de comportamiento creando los archivos con la nomenclatura descrita en la sección 4.4.1.
- **agroeco.mas.code.gen.definirComportamiento.AgenteLocalizado:** Análoga a la extensión anterior pero para la entidad *Agente_Localizado.*
- **agroeco.mas.code.gen.definirComportamiento.RecursoLocalizado:** Ídem para entidad *Recurso_Localizado*.
- **agroecosistemas.mas.code.gen.definirComportamiento.Recurso:** Ídem para entidad *Recurso*.

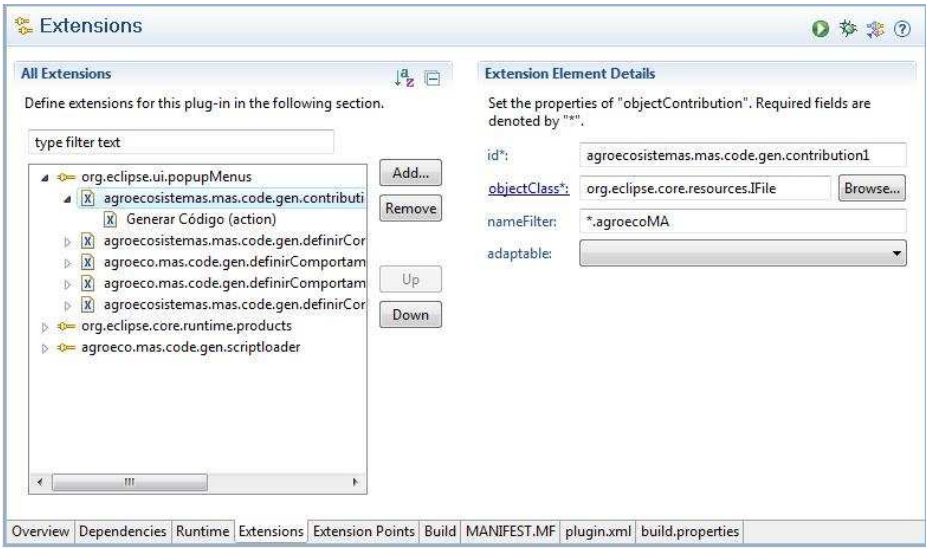

Figura 4.7: Extensiones

Capítulo 5

Caso de Estudio

En este capítulo se detalla la implementación de un caso de estudio, permitiendo de esta forma realizar una demostración del funcionamiento de FW4SimSMA.

5.1. Planteo de la realidad a modelar

El caso de estudio se basa en el modelo definido en el documento referenciado en [16]. Este trabajo tiene como objetivo modelar la dinámica de los sistemas agrarios en Uruguay y en base a dicha definición poder comprender y predecir los impactos de los cambios del uso del suelo, dado el gran interés en el estudio de la evolución del uso del suelo principalmente en los últimos años, a consecuencia del cambio vertiginoso generado por la producción de soja y el aumento de la forestación.

Para la definición del modelo se seleccionaron dos grandes subconjuntos de productores: por un lado el productor tradicional agrícola-ganadero que integra en su sistema de producción la rotación de cultivos, pasturas y la agricultura con producción animal (vacunos de carne, lechería u ovinos) y por otro lado se consideran los nuevos empresarios agrícolas que basan su sistema productivo en la agricultura continua sobre campos tomados en arrendamiento. El modelo simula el comportamiento de dichos productores, su interacción así como los efectos producidos por factores internos o externos como por ejemplo el aumento de la demanda de la soja lo que repercute en su precio.

Para el modelado se establecieron los siguientes supuestos:

- Los productores pueden dar en alquiler una o más de sus parcelas.
- En las parcelas arrendadas siempre se realiza agricultura continua (soja).
- Los productores no hacen un uso único de sus parcelas, ya que cuando alquilan (dan en arriendo) el uso de la parcela alquilada no tiene por qué coincidir con lo que él venía haciendo.
- No existe comunicación entre los productores tradicionales.

5.2. Implementación del modelo

La Figura 5.1 describe los pasos realizados para la implementación del caso de estudio.

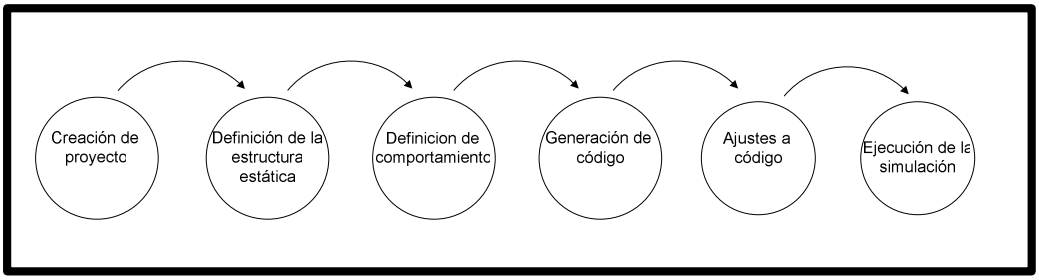

Figura 5.1: Pasos para la implementación de un caso de estudio

A continuación se describen los pasos ilustrados anteriormente.

5.2.1. Creación del proyecto

El primer paso consistió en la creación de un proyecto Java estándar llamado CASO-ESTUDIO en el IDE Eclipse.

5.2.2. Definición de la estructura estática

Este paso consistió en crear un diagrama de estructura estática para poder definir las entidades que componen el modelo llamado Dinámica Parcelaria.
En la Figura 5.2 se muestra el diagrama de estructura estática definido en el documento de referencia [16] al cual nos referiremos como diagrama original. En este diagrama se definen las entidades *Productor, Familiar y Empresarial,* donde las dos primeras representan a los productores tradicionales y la tercera a los nuevos agricultores. Estas entidades tienen asignadas una cantidad determinada de parcelas de tierra, representadas por la entidad *Parcela*, y dependiendo de la actividad del productor se determina su uso, el cual está representado por la entidad *Uso*.
En la Figura 5.3 se presenta la definición del diagrama de estructura estática definido en FW4SimSMA.

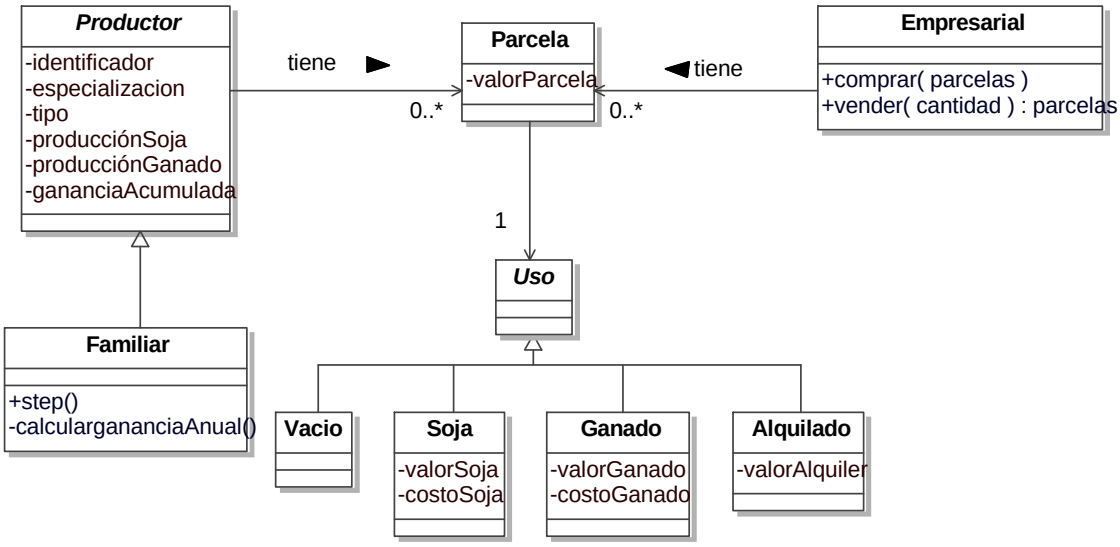

Figura 5.2: Diagrama de estructura estática original

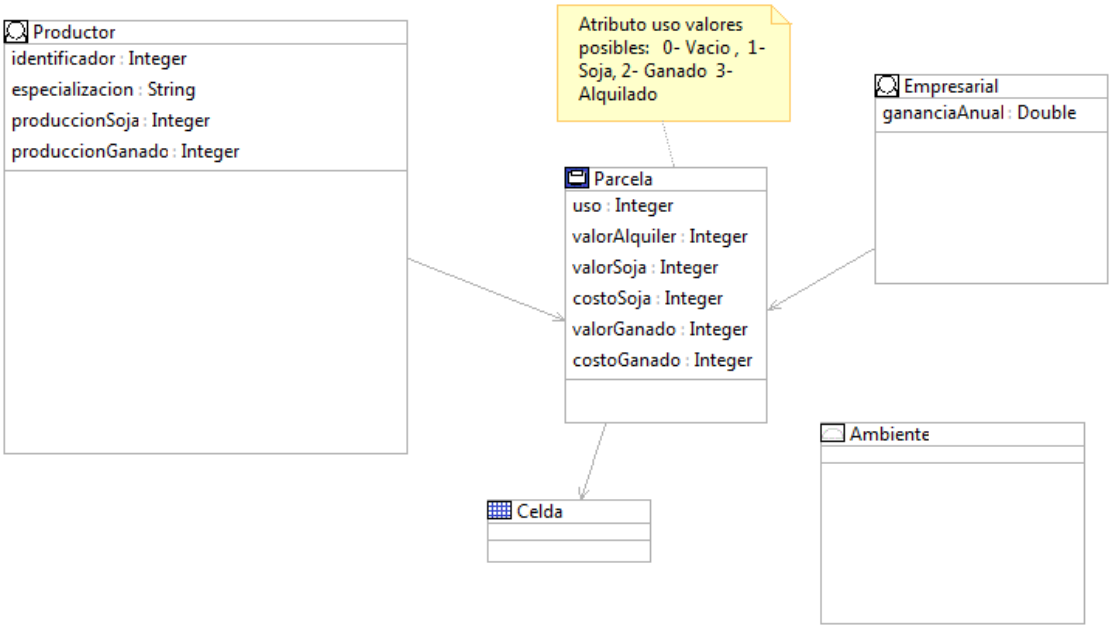

Figura 5.3: Diagrama de estructura estática desarrollado en FW4SimSMA

Una de las simplificaciones realizadas para la implementación del caso de estudio fue solo considerar a los productores tradicionales, llamados simplemente *Productor* y los nuevos productores llamados *Empresarial*. También se debe de aclarar que dada la imposibilidad para representar herencia en FW4SimSMA, se realizaron algunas adaptaciones al diagrama original. Por ejemplo las entidades *Productor* y *Familiar* se unificaron en una sola entidad llamada *Productor*, como se ilustra en la Figura 5.3.

5.2.3. Definición de comportamiento

Se realizó la especificación del comportamiento para las entidades *Productor* y *Empresarial*.

Definición de comportamiento del agente Productor

El comportamiento del agente *Productor* consiste básicamente en determinar cuál será su actividad el próximo año, en función de la que tenga mayor margen bruto. Dependiendo de la evolución de su ganancia va tomando la decisión de comprar, vender, alquilar o desalquilar sus parcelas.

La Figura 5.4 presenta el diagrama de actividad definido en el documento de referencia [16] el cual nos referiremos como diagrama original. En este diagrama se especifica el comportamiento del agente *Productor*. Al final de cada año el productor calcula su ganancia; ve la ganancia acumulada que posee, y en función de ello decide si sigue produciendo lo mismo, si puede comprar nuevas parcelas, si puede desarrendar parcelas que tenía alquiladas o si debe alquilar o vender su tierra. Esta decisión es tomada en función del monto de ganancia acumulada que posee. Una vez definida esta situación define qué actividad productiva realizar.
En la Figura 5.5 se ilustra el diagrama de actividad para el Productor definido en FW4SimSMA.

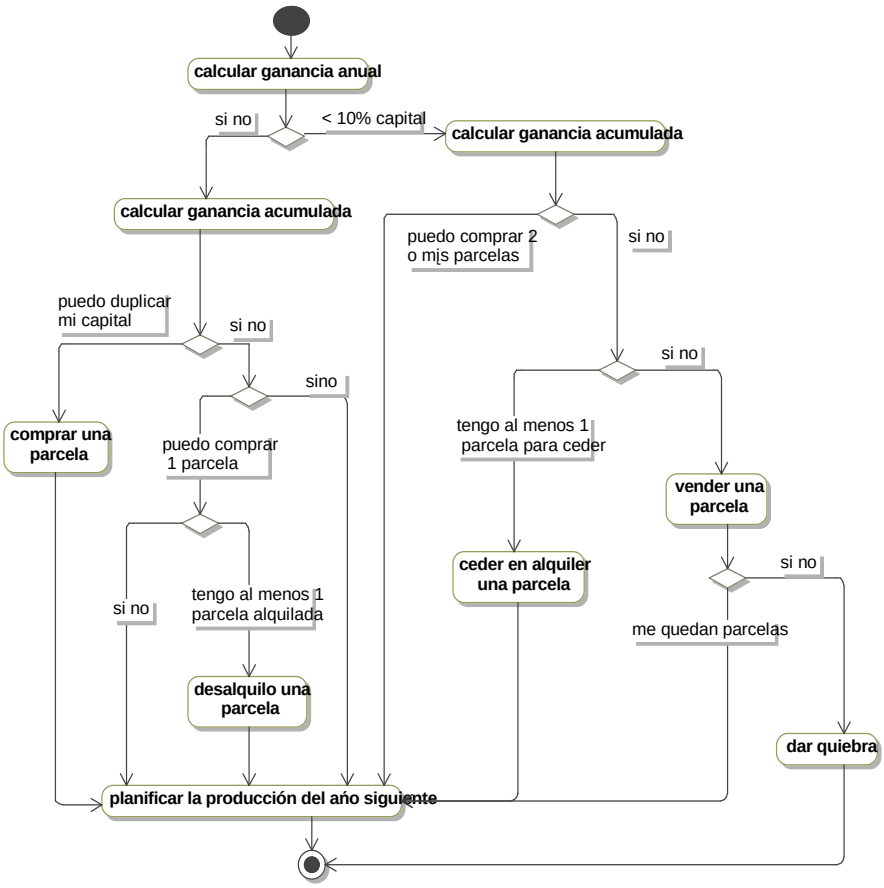

Figura 5.4: Diagrama de actividad original para un Productor

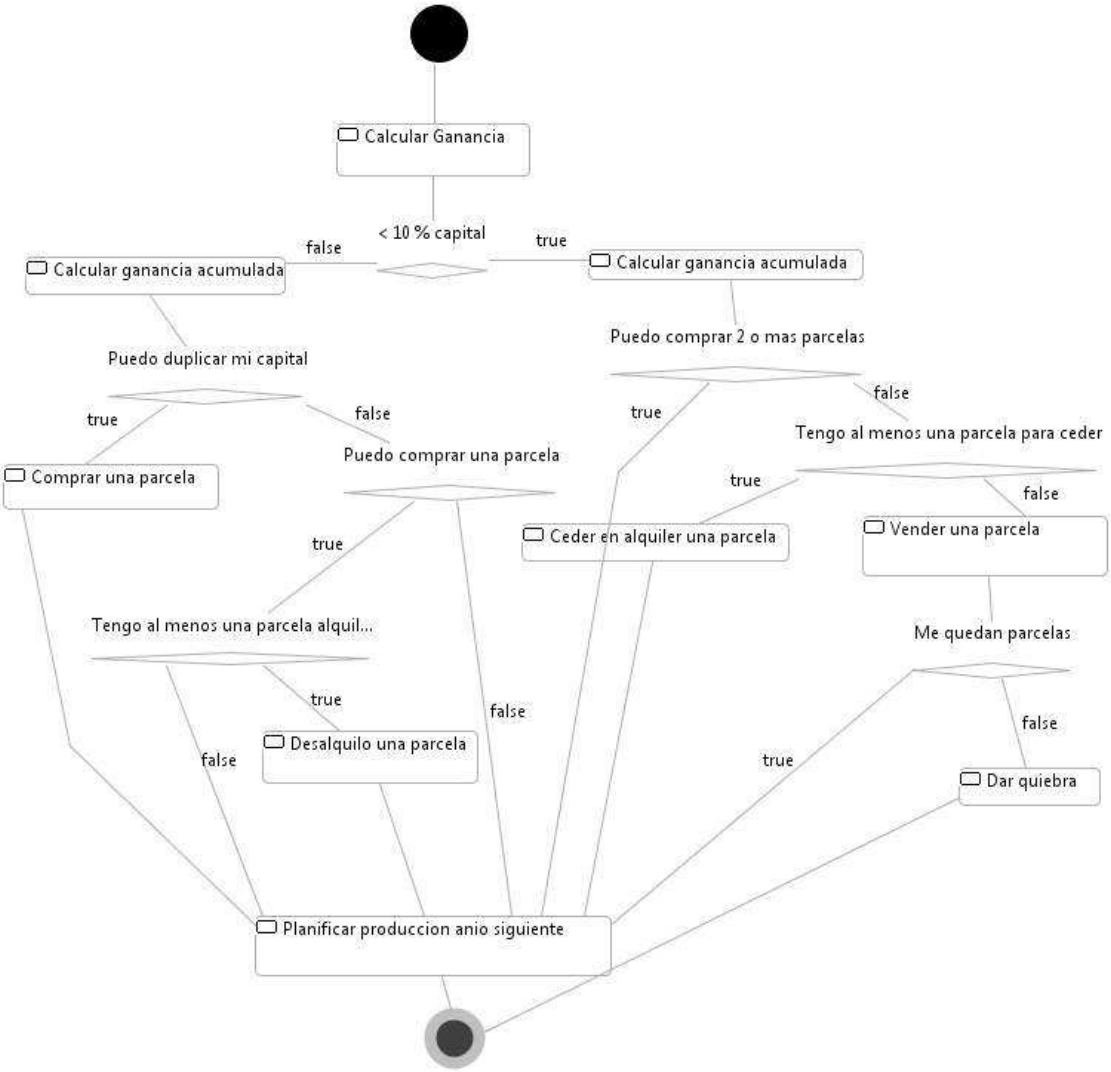

Figura 5.5: Diagrama de actividad para un Productor desarrollado en FW4SimSMA

Como se puede observar, el diagrama de actividad definido en FW4SimSMA respeta en su totalidad al diagrama de actividad original.

Definición del comportamiento del agente Empresarial

Dado que en el documento de referencia [16] para el caso de estudio, no hay especificado un comportamiento para la entidad *Empresarial*, previa consulta con uno de los autores del documento de referencia se procedió con la definición de uno que se detalla a continuación.

El agente al final de cada año calcula su ganancia anual y si ésta fue superior al diez por ciento de su capital trata de alquilar nuevas parcelas, contando con un monto del diez por ciento de su ganancia actual, de lo contrario desalquila una parcela para reducir costos.

La Figura 5.6 presenta el diagrama de actividad para el *Empresarial* definido en FW4SimSMA.

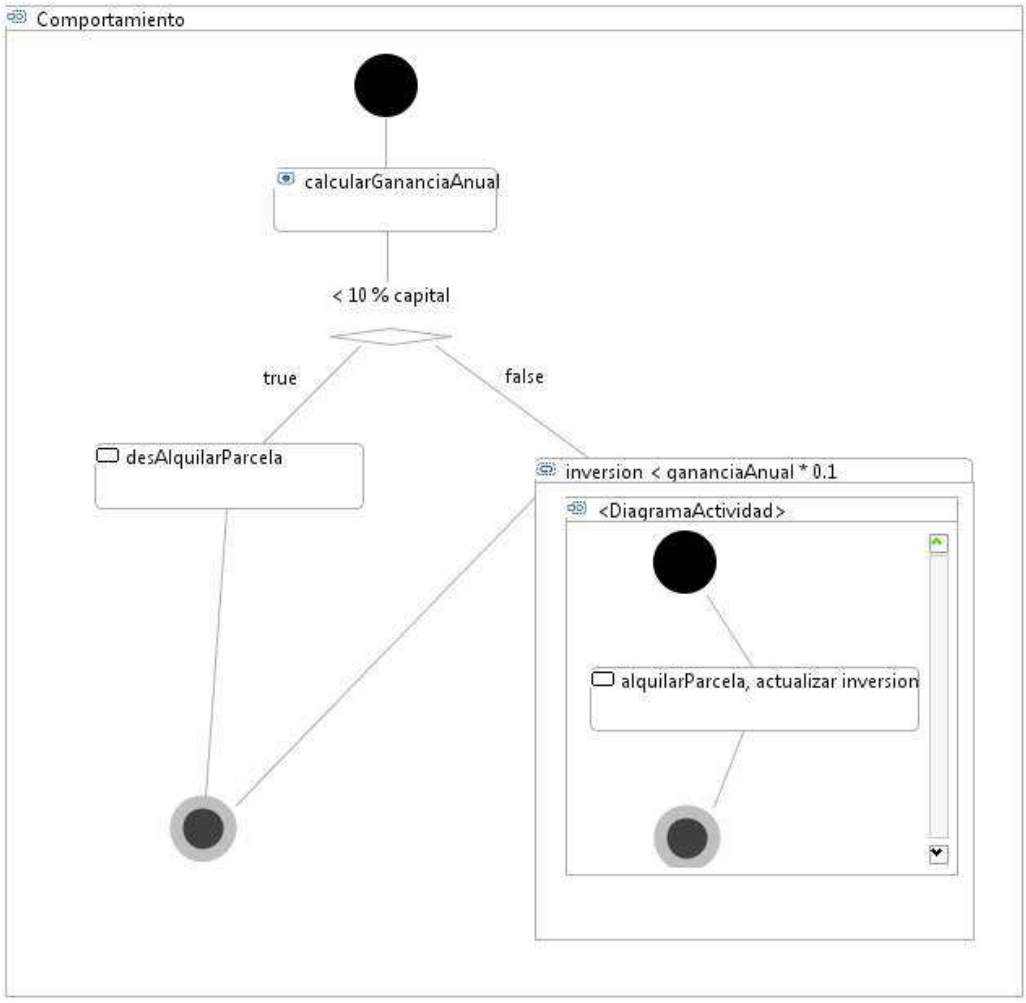

Figura 5.6: Diagrama de actividad para el Empresarial desarrollado en FW4SimSMA

La construcción identificada por la expresión "inversión < gananciaAnual * 0.1" corresponde a la especificación de un ciclo, donde la expresión anterior es su condición booleana. El cuerpo de un ciclo se representa con un diagrama de actividad, permitiendo de esta forma definir dentro de un ciclo las mismas construcciones que para un diagrama de actividad.
En este caso mientras que la inversión sea menor al diez por ciento de la ganancia anual el agente *Empresarial* trata de alquilar más parcelas.

5.2.4. Generación de código

Se procedió a realizar la generación del código Java el cual contiene las construcciones básicas necesarias para su integración con el framework Ascape, el cual es el responsable de la ejecución de la simulación.

La Figura 5.7 ilustra las clases generadas correspondientes a cada una de las entidades definidas en el diagrama de estructura estática más la clase Model.java, la cual es

tomada por Ascape para ejecutar la simulación. En dicha clase se define la lógica y estructuras que Ascape requiere para la definición de la simulación.
También se aprecian cada uno de los diagramas correspondientes a la especificación del modelo estático y a cada uno de los comportamientos, así como el archivo XML necesario para la generación del código.

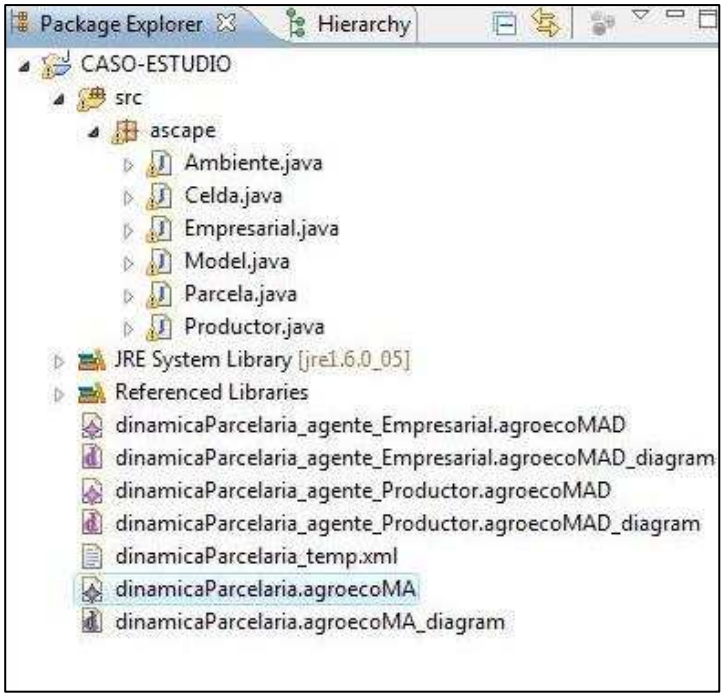

Figura 5.7: Estructura de clases para el proyecto generado

Los archivos .agreoecoMA_diagram y .agreoecoMA corresponden a la definición del diagrama de estructura estática. En el primero se realiza la definición del diagrama de clases (ejemplificado en la Figura 5.3) mientras que el segundo es generado automáticamente y representa con una estructura arborescente basada en XMI el diagrama antes definido.
Los archivos dinamicaParcelaria_agente_Productor.agroecoMAD y dinamicaParcelaria_agente_Empresarial.agroecoMAD corresponden a los diagramas de actividad de los agentes Productor y Empresarial respectivamente, mientras que el archivo dinamicaParcelaria_temp.xml corresponde a la unificación de los archivos anteriores, el cual es entrada al modulo de generación semiautomática de código.

A modo de ilustración se adjunta el código generado para la entidad Empresarial.

```
package ascape;

import org.ascape.model.CellOccupant;

public class Empresarial extends CellOccupant {
   private Double gananciaAnual;

       public void scapeCreated() {
              //Aplico reglas
              getScape().addRule(UPDATE_RULE);
       }

       public void update(){
              this.step();
       }

       /*
        * metodo customizable, se define la logica del agente
        */
       public void step (){
              //calcularGananciaAnual();
              if (true){ //< 10 % capital
                  //desalquilarParcela;

              }else{
                  while (true){
                     //alquilarParcela, actualizar inversion;

                  }
              }
       }

    public void setGananciaAnual(Double gananciaAnual){
       this.gananciaAnual = gananciaAnual;
    }

    public Double getGananciaAnual(){
       return this.gananciaAnual;
    }

}
```

Todos los agentes deben de extender de la construcción *CellOccupant*, la cual es una clase provista por Ascape, que proporciona las construcciones básicas para las entidades que son miembros (ocupantes) de una grilla.
El método *scapeCreated* es invocado automáticamente por Ascape y en él se define la regla de comportamiento a utilizar. En este caso se usa la construcción UPDATE_RULE provista por Ascape mediante la cual se define el método a invocarse en cada paso de la simulación, dicho método es el *update*. Este método llama al método *step* el cual se utiliza para la definición del comportamiento.

Dado que el código generado por FW4SimSMA es un esqueleto, es necesario completarlo manualmente para obtener el comportamiento requerido.
A continuación se adjunta el código completo del método *step* de la entidad *Empresarial*.

```
/*
 * metodo customizable, se define la logica del agente
 */
public void step (){
      if (misParcelas.isEmpty()){
            //Se obtienen las coordenadas de las parcelas
           misParcelas = model.asignarParcelas();
      }
      //calcularGananciaAnual();
      calcularGananciaAnual();
      calcularCapitalTierras();
      double ganancia = gananciaAnual;
      double capital =  capitalTierras;
      if (ganancia < (0.1  * capital)){ //< 10 % capital
          //desAlquilarParcela;
            desAlquilarParcela();
      }else{
            double inversion = 0;
          while (inversion < ganancia){
            //alquilarParcela, actualizar inversion;
            Parcela parcela = alquilar();
            if (parcela != null){
                  inversion = inversion +
                parcela.getValorAlquiler();
            }else{

                  break;
            }

          }
      }
}
```

El primer paso consiste en asignarle parcelas (si es que no tenia asignadas), luego se calcula la ganancia anual, el capital y por último se procede a completar la implementación de la lógica correspondiente a su comportamiento.

5.2.5. Ejecución de la simulación

Luego de completar el código generado, lo que involucra la definición de las características particulares de la interface gráfica, como por ejemplo el color con el cual se representa el uso de cada parcela, así como la inicialización de los atributos específicos, se procede a la ejecución de la simulación. Para ello se debe ejecutar la clase principal org.ascape.runtime.swing.SwingRunner de ASCAPE, a la cual se le pasará como argumento el modelo definido, en este caso ascape.Model.

Al ejecutar la simulación se presenta la interface de usuario de ASCAPE (ver Figura 5.8) y luego la grilla en la cual se visualiza el uso de cada parcela.
Las parcelas rojas son las utilizadas para ganadería, las verdes para soja, las grises son las alquiladas y las blancas las que están sin uso.

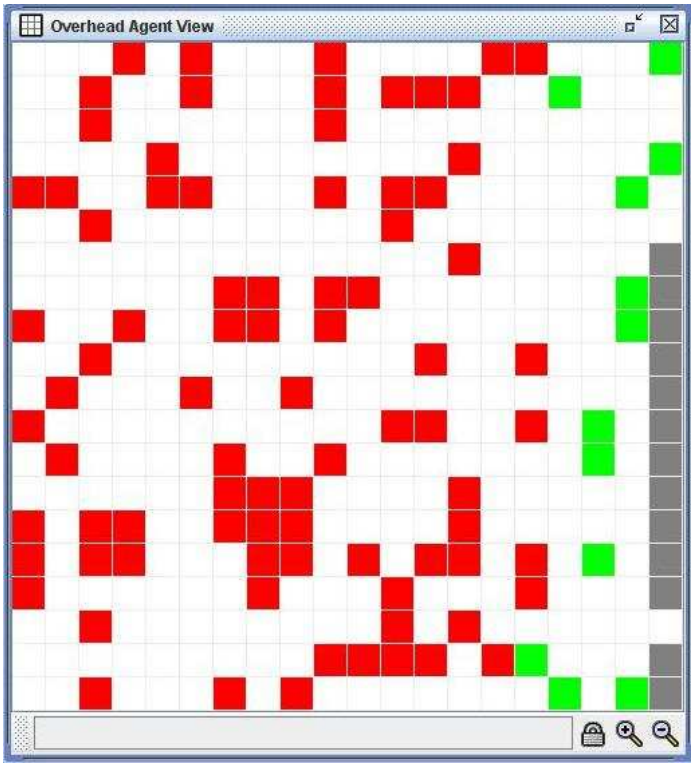

Figura 5.8: Ejecución de la simulación con Ascape

Capítulo 6

Resumen, Conclusión y Trabajo a Futuro

A continuación resumimos y concluimos el trabajo realizado, detallando algunas mejoras que podrán realizarse como extensiones y trabajo a futuro.

6.1. Resumen y conclusiones

El objetivo de este proyecto fue evaluar la factibilidad del desarrollo de un framework de simulación basado en SMA, que mejorara las capacidades de los entornos de simulación existentes. Para cumplir con lo propuesto es que se realizaron un conjunto de tareas.
Inicialmente se analizó el framework Cormas extrayendo sus potencialidades y determinando sus defectos, el cual fue tomado como base para la definición de criterios de evaluación para la extensión o creación de un nuevo framework. Luego se realizó el estudio de distintos frameworks preseleccionándose de un conjunto de veintisiete a Ascape, MASON, SeSAm, Mimosa y RePAST. Para estos frameworks se profundizó su investigación realizando aplicaciones prácticas, que permitieron adquirir un mayor conocimiento de los mismos y poder determinar cual se adecuaba a las necesidades del presente proyecto. Debido a que todos los frameworks considerados cuentan con un entorno de simulación similar, en lo que refiere a funcionalidades y capacidades, nos enfocamos en el modo de especificación y representación del modelo y no tanto en el entorno de simulación, siendo Ascape el framework seleccionado.
Si bien Ascape fue elegido como framework de simulación basado en SMA, el mismo no presenta un editor gráfico para el desarrollo de modelos debiéndose realizar todo el modelado programáticamente. Por tal motivo se decidió utilizarlo como framework para el desarrollo de simulaciones de SMA extendiendo el framework y desarrollando la implementación de un editor gráfico que interopere con Ascape y permita mediante el modelado gráfico, la generación semiautomática del código necesario para la ejecución de una simulación (FW4SimSMA).

FW4SimSMA debía ejecutarse sobre la plataforma Eclipse. Se escogieron los frameworks para eclipse EMF y GMF para el desarrollo del modelo y el editor gráfico respectivamente.
En el modelado se definieron distintos tipos de entidades algunas de ellas para la especificación de diagramas de estructura estática (particularmente diagramas de clase) las cuales se basaron en AML y otras para la especificación de diagramas de comportamiento (particularmente diagramas de actividad) basadas en UML.
Una vez definido el modelo, el mismo fue importado a GMF y se definió el editor visual. En el desarrollo del editor se realizaron dos editores gráficos, uno para el modelado de los diagramas de estructura y otro para los diagramas de comportamiento, permitiendo su vinculación gráfica, para así poder definir comportamiento a los elementos de nuestro dominio.

Si bien GMF es un framework cuyo uso es extendido para el desarrollo de editores gráficos a partir de modelos, presentó varias dificultades, que detallaremos a continuación:

- La generación semiautomática de las propiedades detalladas en los modelos no resultó ser la especificada, por lo que implicó modificar y en ocasiones rehacer algunos de los procesos que hacen al editor (por ejemplo los mapeos).

- Para la validación de los modelos gráficos se definieron reglas de validación (*Rule validators*), en las cuales se especificaron las restricciones de los modelos, pero el lenguaje de especificación no era OCL, sino una especificación híbrida que toma a OCL como base. Esto dificultó la especificación de las reglas, además de no disponer de un validador sintáctico para las expresiones, lo cual entorpeció aun más la definición de las reglas, en donde en su mayoría la corrección fue a base de ensayo y error.

- La correcta definición de los procesos y modelos mediante el uso del framework no resultó muy flexible, ni amigable. Si bien cada proceso provee de una validación, éstas no resultan muy descriptivas y la detección de errores no es una tarea simple.

- Podemos encontrar diversos tutoriales y documentación para el desarrollo de editores con GMF, pero toda la documentación redunda en los mismos ejemplos y básicamente el mismo nivel de detalle, por lo cual no se encontró un tutorial que cubriera las distintas propiedades y profundizara en el uso de GMF para el correcto desarrollo de nuestro editor, debiendo muchas veces dedicar muchas horas de investigación para la implementación del mismo.

Luego de definido el modelo y el editor gráfico, fue necesario realizar la generación de código Java, a partir de los modelos especificados por el usuario. Para realizar esta tarea se utilizó Acceleo como framework para la generación de código.
Al estar Acceleo basado en EMF resultó ser compatible con cualquier herramienta basada en este framework, lo que permitió su compatibilidad e interoperabilidad con GMF.
Mediante el uso de Acceleo se pudo realizar la generación semiautomática de código, necesario para interoperar con el framework de simulación seleccionado (Ascape).
Un aspecto relevante que se evidencia al momento de completar el código generado para poder realizar la ejecución de una simulación, es que el usuario deberá tener profundos conocimientos del framework Ascape, para ultimar los detalles de implementación necesarios para el modelo específico, como pueden ser por ejemplo la representación gráfica de la información a evaluar. Por tanto, el uso de Ascape resultó complejo, ya que si bien permite modelar SMA en el dominio de agro-ecosistemas, dicho modelado no es trivial, debido a que no es un framework específico para la realización de simulaciones de SMA en el dominio definido. Por este motivo para poder generar código semiautomático, interoperar y generar una simulación se debieron realizar varios ajustes para que FW4SimSMA se comportara según lo requerido.
Finalmente, como resultado de este trabajo se construyó FW4SimSMA para la plataforma Eclipse que permite a un usuario crear un modelo basado en SMA, definiendo estructura y comportamiento a través de un editor gráfico y en base al

modelo definido generar semiautomáticamente el código Java necesario para la ejecución de una simulación.
FW4SimSMA provee una interface amigable para cada una de las etapas del modelado, facilitándole al usuario el acceso a las construcciones habilitadas para cada uno de los diagramas (estructura y comportamiento).

En base al trabajo realizado concluimos la factibilidad del desarrollo de un framework, cuya implementación se basa en la interoperabilidad de los frameworks EMF, GMF, Acceleo y Ascape, los cuales permiten la integración con Eclipse en el desarrollo de simulaciones basadas en SMA para el contexto de agro-ecosistemas.

6.2. Trabajo futuro

Si bien en este proyecto se cumplió con los objetivos establecidos, se detectaron varios puntos donde la continuidad, extensión, y profundización de los mismos brindarían una mejora al trabajo realizado. A continuación detallamos los puntos a mejorar, enmarcados como trabajo a futuro.

Framework de simulación
La selección del framework Ascape como framework de simulación si bien se ajusta a los criterios establecidos, no es un framework específico para la realización de simulaciones de SMA en el dominio de agro-ecosistemas (por ej.: no contiene construcciones para un ambiente). Por tanto, sería deseable el desarrollo de un framework para el desarrollo de simulaciones específicas para dicho dominio.

Extensión de diagramas para el modelado
Mediante la especificación de diagramas basados en UML y AML, la realidad es modelada utilizando el enfoque de SMA, particularmente diagramas de clases y de actividad. La extensión de los diagramas, incluyendo otros nuevos podría ser de utilidad, para el modelado de aspectos no considerados. Por ejemplo: mediante la inclusión de diagramas de comunicación, se podría modelar de mejor forma la ejecución de la simulación.

Mejoras en la representación gráfica de los diagramas de clases
Los diagramas de estructura además de las clases específicas del dominio, permiten modelar nombres (de clases), atributos, tipos de atributos, y relaciones entre las clases. Sería deseable poder especificar en el modelado, por ejemplo:

- Operaciones con sus respectivos parámetros de entrada, tipos y retorno.
- Herencia, para los agentes y recursos.

Mejoras en la representación gráfica de los diagramas de actividad
Los diagramas de actividad, permiten el modelado de *Condiciones*, *Métodos*, *Actividades*, *Ciclos*, elementos de *Inicio* y *Fin* de actividad. Se podría extender el alcance del elemento *Condición*. Actualmente se expresan como texto que representan condiciones booleanas, no teniendo ningún tipo de validación. Por tanto, sería deseable en este caso la construcción de condiciones booleanas, el uso de operadores (and, or, not, <, >, =) y la realización de una validación de la sintaxis de la condición, de igual forma también debería realizarse la validación y construcción de una condición para un *Ciclo*.

Mejoras en funcionalidades del framework
Ascape cuenta con la funcionalidad para realizar análisis estadísticos de las propiedades contenidas en una simulación. Como trabajo futuro podría realizarse gráficamente la definición de estadísticas para el análisis y monitoreo de aquellas propiedades (por ej.: atributos de agentes) para las cuales se quiera estudiar su evolución en el tiempo.

Restricciones OCL en GMF
Si bien se definieron un conjunto de restricciones basadas en OCL para los modelos, quedó pendiente la validación de una restricción la cual indica que no puede haber conexiones entre elementos de distintos diagramas de actividad. La falta de realización de la misma, se debió a que la especificación de las restricciones se realizan en un lenguaje basado en OCL, que no permite validación sintáctica a la hora de su realización, por tanto si se toma a GMF como framework para el desarrollo de editores gráficos basado en modelos debería profundizarse en el estudio y especificación de restricciones para los modelos.

Generador de Código
Acceleo como plugin para la generación de código nos permitió la generación semiautomática de código Java. Como trabajo futuro podría enriquecerse el código generado, por ejemplo mediante la definición de templates más completos o mediante diagramas más ricos respecto a la incorporación de nuevas construcciones. De forma tal de minimizar la tarea de un programador y facilitar la realización de una simulación.

Bibliografía

[01] Wooldridge, M. Website, 2008. Lecture Notes on Introduction to Multiagent Systems Course based on based on [Wooldridge, 2002]. http://www.csc.liv.ac.uk/~mjw/pubs/imas/teaching.html Visited: August 2008

[02] UML. http://www.uml.org/

[03] Cormas. http://cormas.cirad.fr/

[04] Eclipse IDE. http://www.eclipse.org/

[05] Java. http://java.sun.com/

[06] Martin Fowler. http://www.martinfowler.com/bliki/DomainSpecificLanguage.html

[07] Bousquet, F., Le Page, 2004. Multi-agent simulations and ecosystem management: a review. Ecological Modeling 176:313-332.

[08] Ferber, J. 1999. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley, Reading, MA.

[09] Wooldridge, M., Jennings, N. R. 1995. Intelligent agents: theory and practice. The Knowledge Engineering Review.

[10] Mohammad S Obaidat and Georgios I Papadimitriou, 2003. Applied System Simulation: Methodologies and Applications.

[11] R.E. Shannon1 Shannon, Robert; Johannes, James D. (1976). «Systems simulation: the art and science». IEEE Transactions on Systems, Man and Cybernetics 6(10). pp. 723-724.

[12] Robert Tobias and Carole Hofmann, 2004. Evaluation of free Java-libraries for social-scientific agent based Simulation. Journal of Artificial Societies and Social Simulation vol. 7, no. 1.

[13] Steven F. Railsback, Steven L. Lytinen and Stephen K. Jackson. Agent-based Simulation Platforms: Review and Development Recommendations.

[14] Maria Bruno Marietto1, Nuno David, Jaime Simão Sichman and Helder Coelho. Requirements Analysis of Agent-Based Simulation Platforms: State of the Art and New Prospects.

[15] Russell K. Standish, School of Mathematics and Statistic, University of New South Wales. Agent Based Modelling Frameworks.

[16] Jorge Corral, Pedro Arbeletche, Julio César Burges, Hermes Morales, Guadalupe Continanza, Jorge Couderc, Virginia Courdin, and Pierre Bommel. Estrategias agrícolas ganaderas y paisajes: El uso de sistemas multiagentes para relacionarlas.

[17] Gliessman, S. R. 1997. Agroecology: ecological processes in sustainable agriculture. Ann Arbor Press Hecht, S. 1987. The Evolution of Agricultural Thought. .

[18] Marco A. Janssen, USA March 2005. School of Informatics & Center for the Study of Institutions, Population and Environmental Change, Indiana University –Bloomington. Agent-Based Modelling.

[19] Aml - Whitestein Technologies. http://www.whitestein.com/

[20] Tulio Jose Marchetti, Alejandro Javier Garcia, 2003. Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA), Departamento de Ciencias e Ingeniería de la Computación – Universidad Nacional del Sur. Evaluación de Frameworks para el Desarrollo de Sistemas MultiAgente.

[21] Horacio Blanco, Diego Salido, Junio de 2009, Montevideo – Uruguay. Informe de Proyecto de Grado, Lenguajes específicos de dominio para simulación multiagente,

[22] GREEN del CIRAD. http://www.cirad.fr/ur/green_en

[23] MASON. http://cs.gmu.edu/~eclab/projects/mason/

[24] Mimosa. http://mimosa.sourceforge.net/

[25] Repast. http://repast.sourceforge.net/

[26] SeSAm. http://ki.informatik.uni-wuerzburg.de/~sesam/

[27] Ascape. http://ascape.sourceforge.net

[28] EMF. http://www.eclipse.org/modeling/emf/

[29] GMF. http://www.eclipse.org/modeling/gmp/

[30] Acceleo. http://www.acceleo.org/pages/home/en

[31] XMI. http://www.omg.org/technology/documents/formal/xmi.htm

[32] GEF. http://www.eclipse.org/gef/

[33] OCL. http://www.omg.org/technology/documents/formal/ocl.htm

[34] Licencias de Software. http://www.gnu.org/licenses/license-list.html

[35] AgentSheets. http://www.agentsheets.com/

[36] AgentTool. http://agenttool.cis.ksu.edu/

[37] Breve. http://www.spiderland.org/

[38] ECHO. http://www.santafe.edu/~pth/echo/

[39] Ingenias. http://grasia.fdi.ucm.es/main/?q=en/node/127

[40] Jack. http://agent-software.com.au/products/jack/index.html

[41] JADE. http://jade.tilab.com/

[42] FIPA. http://www.fipa.org/

[43] MadKit. http://www.madkit.org/

[44] MAGSY. http://www-ags.dfki.uni-sb.de/~kuf/magsy.html

[45] MAML. http://www.maml.hu/

[46] MIMOSE. http://www.uni-koblenz.de/~moeh/projekte/mimose.html

[47] NetLogo. http://ccl.northwestern.edu/netlogo/

[48] PS-i. http://ps-i.sourceforge.net/

[49] SimAgent. http://www.cs.bham.ac.uk/research/projects/poplog/packages/simagent.html

[50] SimPack. http://www.cis.ufl.edu/~fishwick/simpack/simpack.html

[51] StarLogo. http://education.mit.edu/starlogo/

[52] SugarScape. http://sugarscape.sourceforge.net/

[53] Swarm. http://www.swarm.org/

[54] TeamBots. http://www.teambots.org/

[55] VSEit. http://www.vseit.de/

[56] Zeus. http://sourceforge.net/projects/zeusagent

[57] Eclipse Ganymede : http://www.eclipse.org/ganymede/

Glosario

Acceleo	Framework open source basado en los estándares, que permite la generación de código a partir de modelos UML o EMF.
Ascape	Framework para el desarrollo y la exploración de propósito general basada en modelos de agentes.
AML	*Agent Modeling Lenguje*. Es un lenguaje semi-formal para el modelado visual, que permite especificar modelos y documentación de los sistemas que incorporan conceptos de SMA.
Annotation	Especie de meta-tags que se agregan al código y se aplican a declaraciones de paquetes, tipos, constructores, métodos, atributos, parámetros y variables.
API	*Application Programming Interface*. Grupo de rutinas (conformando una interfaz) que provee un sistema operativo, una aplicación o una biblioteca, que definen cómo invocar desde un programa un servicio que éstos prestan. En otras palabras, una API representa un interfaz de comunicación entre componentes de software.
Applets	Componente de software (que suele ser pequeño) escrito en un lenguaje de programación (como Java), que se ejecuta bajo el control de una aplicación más grande que lo contiene (como un navegador web).
Bugs	Defecto en un software o un hardware que no ha sido descubierto por los creadores o diseñadores de los mismos.
CORMAS	Framework para simulaciones Multi-Agente, enfocado hacia los modelos de administración de recursos naturales renovables.
CVS	*Concurrent Version System*. El CVS es un modo de administrar versiones de archivos que permite trabajar fácilmente a más de un desarrollador en el mismo proyecto.
Delay	Tiempo que transcurre entre el inicio de una transacción y la llegada de su respuesta.
Eclipse	Se trata de un entorno de desarrollo integrado (IDE). Es una potente y completa framework de programación, desarrollo y compilación.
EMF	*Eclipse Modeling Framework*. Es un framework para el entorno Eclipse que provee facilidades para la definición de modelos.
FAQ	*Frequently Asked Questions*. Sección que ofrece una recopilación de las preguntas y respuestas más solicitadas por los visitantes o usuarios de un sitio Web. Aclaran dudas de forma rápida, sin leer grandes textos.
Framework	Framework, entorno, marco de trabajo. Desde el punto de vista del desarrollo de software, un framework es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado.

GMF	*Graphical Modeling Framework*. Es un framework para el entorno Eclipse que permite a partir de modelos, generar automáticamente editores gráficos como plugins para Eclipse.
Groovy	Lenguaje de programación orientado a objetos implementado sobre la plataforma Java, el cual proporciona en muchos casos velocidad y simplicidad, se puede considerar como un complemento a Java.
IDE	*Integrated Development Environment*. Aplicación compuesta por un conjunto de herramientas útiles para un programador. Un entorno IDE puede ser exclusivo para un lenguaje de programación o bien, poder utilizarse para varios. Suele consistir de un editor de código, un compilador, un debugger y un constructor de interfaz gráfica GUI, entre otros.
JAR	*Java Archive*. Es un tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje Java, se facilita la distribución e instalación de una gran variedad de archivos, applets, video, sonido, imágenes, texto, etc.
Javadoc	Es una utilidad de Sun Microsystems para la generación de documentación de APIs en formato HTML a partir de código fuente Java. Javadoc es el estándar de la industria para documentar clases de Java.
JSP	*Java Server Pages* (JSP) es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.
Kernel	Parte esencial de un sistema que provee los servicios más básicos del mismo.
Log	Registro oficial de eventos durante un periodo de tiempo en particular. Un Log es usado para registrar datos o información sobre quién, que, cuando, donde y porque un evento ocurre para un dispositivo en particular o aplicación.
MAS	*Multi Agents Systems.* Conjunto de Agentes que interactúan y trabajan conjuntamente de forma colaborativa, para resolver problemas que superan las capacidades individuales o conocimiento de cada uno de ellos.
MaSE	*Multiagent System Engineering.* Metodología completa basada en el ciclo de vida clásico del software, con un entorno propio de desarrollo para analizar, diseñar y construir sistemas Multi-Agente heterogéneos.
OCL	*Object Constrain Lenguaje*. Es un lenguaje formal para la descripción textual y precisa de restricciones que se aplican a los modelos UML.
Ontología	Conjunto de información que es útil en el dominio de un problema. Los Agentes intercambian información según el formato de los hechos especificados en la ontología. La ontología comprende: esquemas de la estructura de los predicados, acciones de los Agentes y conceptos relevantes al dominio.
Plugin	Programa que puede anexarse a otro para aumentar sus funcionalidades (generalmente sin afectar otras funciones ni afectar la

	aplicación principal). No se trata de un parche ni de una actualización, es un módulo aparte que se incluye opcionalmente en una aplicación.
QuickTime	Conjunto de bibliotecas desarrolladas por Apple. Se trata de una completa aplicación multimedia para reproducir o transmitir audio y video por Internet.
Release	Nueva versión de una aplicación informática.
Schedule	Es un plan que define la secuencia y ubicación temporal de las operaciones necesarias para llevar a cabo una tarea.
Script	Grupo de lenguajes de programación que son típicamente interpretados y pueden ser tipeados directamente desde el teclado. Son un conjunto de instrucciones generalmente almacenadas en un archivo de texto que deben ser interpretados línea a línea en tiempo real para su ejecución, se distinguen de los programas, pues deben ser convertidos a un archivo binario ejecutable para correrlos.
Serialización	Consiste en un proceso de codificación de un objeto en un medio de almacenamiento con el fin de transmitirlo. La serialización es un mecanismo ampliamente usado para transportar objetos a través de una red, para hacer persistente un objeto en un archivo o base de datos, o para distribuir objetos idénticos a varias aplicaciones o localizaciones.
Snapshot	Refiere al estado de un sistema en un determinado instante de tiempo.
Time-step	Caso particular de eventos discretos, donde el modelo se ejecuta cada cierta cantidad fija de tiempo.
Tips	Palabra inglesa utilizada para definir a un conjunto de consejos o recomendaciones.
UML	*Unified Modeling Language*. UML es un lenguaje de modelado de sistemas de software. Se trata de un lenguaje gráfico para construir, documentar, visualizar y especificar un sistema de software.
URL	*Uniform Resource Locator*. Forma de organizar la información en la web. Una URL es una dirección que permite acceder a un archivo o recurso. Se trata de una cadena de caracteres que identifica cada recurso disponible en la W3C.
Wiki	Todo sitio Web en donde colaboran múltiples autores. En estos lugares cualquiera puede editar su contenido generalmente utilizando un navegador Web.
Wizard	Aplicación al servicio del usuario (especialmente inexpertos) que generalmente abrevia los pasos a seguir para realizar una tarea, dando una explicación de cómo realizarla.
XMI	*XML Metadata Interchange.* XMI es el nombre que recibe el estándar para el intercambio de metamodelos usando XML.

XML	*Extensible Markup Language*. Desarrollado por el World Wide Web Consortium (W3C). Su objetivo es conseguir páginas Web más semánticas. XML separa la estructura del contenido y permite el desarrollo de vocabularios modulares. Se trata de un formato abierto.

Apéndice A
Estudio extendido de los Frameworks seleccionados

En este apéndice se detalla completamente el estudio realizado para los frameworks seleccionados en el capítulo 3.3.

A. 1. Investigación – MASON (Stands for Multi-Agent Simulator Of Neighborhoods)

MASON [23] es un framework de simulación para SMA basado en eventos discretos. Brinda la posibilidad de serializar la simulación pudiendo recuperarla y ejecutarla en cualquier momento. Puede ser configurado para que se garantice duplicación, lo que significa que los mismos parámetros de simulación producirán los mismos resultados, sin importar la plataforma. Provee varias opciones para la visualización de la simulación y estudio de resultados, por ejemplo la simulación se puede visualizar en 2D o 3D, se pueden obtener gráficos de las variables requeridas, tomar capturas de pantalla y generar películas, entre otras.

Criterios Generales

Actualización y Mantenimiento: Es un framework mantenido, el lanzamiento de su última versión fue en el año 2008 (release 13).

Documentación: Existe variada documentación orientada a distintos tipos de usuarios o intereses. Por ejemplo se dispone documentación de usuario para el manejo de la interfaz de simulación, documentación para desarrolladores, preguntas frecuentes y tips de uso.

Lenguaje de desarrollo/especificación: Java/Java.

Licencia: Software open source bajo la licencia Académica Libre [34], versión 3.0.
A excepción de los siguientes archivos:

PngEncoder.java	Licencia Artística
MovieEncoder.java	[parcial] Sun de licencia Open Source
WireFrameBoxPortrayal3D.java	[parcial] Sun de licencia Open Source
ToolTipBehavior.java	[parcial] Sun de licencia Open Source
SelectionBehavior.java	[parcial] Sun de licencia Open Source
GullCG.java	Sun de licencia Open Source
MersenneTwisterFast.java	Licencia BSD

Mecanismos de extensión: Dado que MASON está diseñado para utilizarse como una librería de Java, el mecanismo de extensión es el programático, se pueden definir nuevas bibliotecas que utilicen la biblioteca MASON o directamente modificar el código fuente.

Soporte: Se pueden realizar consultas a través de una lista de correo, en los manuales provistos se detalla el mecanismo para realizar la subscripción.

Integración con Eclipse: Como ya hemos mencionado MASON es una librería, por lo tanto se puede incorporar el .jar como una dependencia a proyectos específicos.

Uso: Está pensado para ser usado como una librería, por lo tanto la generación de nuevos modelos requiere que los usuarios posean conocimientos de programación (específicamente Java).

Diseño del framework

Organización del código y estructura interna: Al inspeccionar el código se observó que las clases principales se encuentran comentadas y la implementación sigue un *code style* unificado, por el contrario a las clases que conforman los ejemplos incluidos en la distribución. Se dispone de un completo tutorial de clases, así como de una API. Las clases se agrupan en paquetes dependiendo de sus responsabilidades inherentes, por ejemplo se tiene la siguientes división *app* (aplicaciones de ejemplo), *display* (interfaz 2d), *display3d* (interfaz 3d), *engine* (lógica de Agentes), *field* (lógica de espacio) y *útil* (algoritmos o estructuras auxiliares).

Modelado

Especificación del modelo: La definición del modelo que consiste en la creación de agentes, comportamiento, ambiente y recursos se realiza exclusivamente de forma programática.
Para definir un modelo se necesita un *observer* que proporciona la información gráfica a ser visualizada (del agente, paneles de control, parámetros, etc.), un objeto *model* que contiene el modelo real del agente, su comportamiento y la planificación (*schedule*) que debe ser ejecutada en cada ciclo (paso), una o varias colecciones de agentes y uno o varios objetos para representar el *entorno* del agente.

MASON define una clase de tipo *user interface* (UI) que es la responsable de la visualización del modelo en ejecución.
Las clases que contienen los objetos que utilizará el *schedule* deben derivar de *Steppable,* estas deberán ser definidas por el usuario, ya que no se cuenta con clases específicas para la definición de las entidades agente y recurso.

Formato de almacenamiento: El modelo se guarda como clases Java.La ejecución de una simulación puede ser serializada en archivos con extensión ".checkpoint" (estándar de serialización Java).

Entorno de simulación

Definición de una simulación: La simulación se realiza mediante la interfaz que se haya definido para el modelo específico. MASON permite tener múltiples simulaciones corriendo al mismo tiempo.
Brinda una amplia variedad de opciones para la representación del ambiente, aparte de la simple cuadrícula 2D, también permite formas de campo hexagonal y campos continuos. El entorno de simulación proporciona funcionalidades básicas como es el

poder visualizar el estado de un determinado agente así como editar los valores de los parámetros de la simulación.

Manipulación de datos de salida: El entorno de simulación provee varios mecanismos para el análisis de los datos de simulación. Se puede inspeccionar cada agente realizando doble click sobre él, mostrando sus propiedades y valores respectivos. También se pueden generar gráficos definiendo cuales son las propiedades a analizar, en un mismo gráfico se pueden incorporar varias propiedades.

Avance en el tiempo: El avance del tiempo es del tipo time-step.

Tipo de inicialización: Se pueden editar los valores de las distintas variables de la simulación, también se brinda la posibilidad de definir sobre que atributos generar gráficos, tanto de los globales como de los atributos particulares de cada agente.

Uso de una semilla: Desde el entorno de simulación se permite definir o cambiar el valor de una semilla para la generación de una simulación.

Controles de ejecución: En general el entorno de simulación permite ejecutar, pausar y parar la simulación. Si se pausa la simulación se puede avanzar paso a paso. Proporciona algunos seteos generales como por ejemplo insertar *delay* entre cada lapso de tiempo para los casos en que la simulación sea muy rápida, permite incrementar la prioridad del hilo de la simulación y definir la cantidad de pasos a moverse con cada presión sobre el botón Play siempre y cuando la simulación esté pausada.

Acceso a la simulación: Desde el entorno de simulación se puede generar una grabación de la simulación así como tomar imágenes instantáneas. La simulación también puede ser persistida y luego ser reanudada desde un punto de ejecución determinado, no se cuenta con la posibilidad de ser reproducida.

A. 2. Investigación – Mimosa

Mimosa [24] es un framework de modelado y simulación, que cubre desde el proceso de creación de modelos hasta la ejecución de las simulaciones. Es un framework concebido para ser "sucesor" de Cormas, los principales objetivos son los de extender la funcionalidades provistas por Cormas así como la de utilizar nuevas tecnologías en su desarrollo.

Criterios generales

Actualización y Mantenimiento: No es una aplicación mantenida, la última versión fue liberada el 19 de abril del 2006 y se encuentra en versión beta.

Documentación: Se dispone de una escasa documentación la cual básicamente incluye un manual de usuario y una API.

Lenguaje de desarrollo/especificación: Java/Java.

Licencia: LGPL [34].

Mecanismos de extensión: Se puede extender programáticamente.
Soporte: Se pueden realizar consultas a través de una lista de correo.

Integración con eclipse: Puede utilizarse eclipse como IDE, configurándose el CVS para descargar el proyecto. No es necesario hacer uso del IDE para el modelado y ejecución de una simulación ya que esto se hace desde el entorno gráfico provisto.

Uso: Es un framework que provee un entorno gráfico, en la cual se puede definir todo el ciclo de vida de un modelo, desde la creación del modelo conceptual, pasando por la especificación del comportamiento terminando con la ejecución de la simulación y estudio de resultados. Se requieren conocimientos de diagramas UML para la definición de modelos conceptuales así como nociones de programación para poder especificar comportamiento. Su entorno gráfico no es amigable ni intuitivo, puede resultar agobiante al desplegarse tantas ventanas y ser muy confuso a la hora de tratar de definir un modelo.

Diseño del framework

Organización del código y estructura interna: El código se puede obtener del repositorio. Las clases están debidamente comentadas y se cuenta con una completa API. No se encontró un diagrama de arquitectura, ni manuales específicos para desarrolladores que expliquen en detalle el diseño de la aplicación. Se realizo una inspección al código lo cual evidenció una agrupación de clases en paquetes pero sin una división clara respecto a responsabilidades ni funcionalidad.

Modelado

Especificación del modelo: La aplicación provee cuatro ventanas o paneles principales, cada uno de ellos con una función específica:

- *Conceptual Model*: En este panel se define el modelo conceptual, consiste en la especificación del dominio, el cual está conformado por un conjunto de categorías, sus atributos y sus relaciones. También se especifica el tipo de comportamiento o dinámica de cada entidad lo cual se puede realizar por medio de las dinámicas estándar ofrecidas o de forma programática.
- *Concret Model:* Se define un escenario específico para el modelo declarado, determinando la representación gráfica de las entidades y del ambiente.
- *Scheduler*: Panel de control de la simulación.
- *Output*: Panel donde se muestra el log de la simulación así como posibles errores en la ejecución.

Para la especificación del modelo de forma programática se utilizan lenguajes Script. En mimosa no hay un motor de compilación para cada uno de los lenguajes indicados, sino que hay un intérprete de Script para cada uno de ellos, que luego lo traduce en Java.

Lenguajes de Scripting:

- Java
- Scheme
- Python
- Prolog
- Smalltalk

Formato de almacenamiento: Almacena el modelo conceptual, el modelo específico y la simulación en archivos .xml.

Entorno de simulación

Definición de una simulación: Para definir una simulación se debe de guardar un escenario específico desde el panel *Concret Model* con la extensión "sim", que es la extensión que identifica a los .xml de simulación. Luego la simulación se ejecuta desde el panel *Scheduler*.

Manipulación de datos de salida: Los datos de salida de una simulación no se pueden grabar, solo imprimir o salvar la imagen generada.
Brinda la posibilidad, de inspeccionar los agentes y poder observar su estado actual.

Avance en el tiempo: El avance del tiempo es del tipo time-step.

Tipo de inicialización: Desde el entorno de simulación se pueden editar los valores de los parámetros globales definidos en el modelo.

Uso de una semilla: Se puede definir o cambiar el valor de una semilla desde el entorno de simulación.

Controles de ejecución: El entorno de simulación provee seis botones principales de control los cuales permiten inicializar, ejecutar, ejecutar paso a paso, detener, reiniciar y cerrar la simulación.

Acceso a la simulación: No se puede grabar la simulación ni las salidas gráficas, pero se puede visualizar el log y chequear paso a paso la simulación.

A. 3. Investigación – RePAST (Recursive Porous Agent Simulation Toolkit)

Repast Simphony [25] está compuesta por varios módulos que buscan facilitar la creación y uso de modelos basados en agentes. Posee una gran variedad de características, dentro de las cuales se destaca la posibilidad de definir el modelo de forma programática o mediante la creación de diagramas de flujo. Brinda integración con una gran variedad de herramientas externas, lo que posibilita la extensión de las funcionalidades ofrecidas y bibliotecas especializadas que proporcionan algoritmos genéticos, redes neuronales y generación de números aleatorios.

Criterios generales

Actualización y Mantenimiento: Es un framework mantenido, la última actualización se realizó en el 2009.

Documentación: La documentación es completa, está compuesta por documentación técnica, manual de usuario, referencias y foro en línea.

Lenguaje de desarrollo/especificación: Java/Java.

Licencia: BSD [34].

Mecanismos de extensión: Hereda un diseño que contribuye a su extensibilidad, el mecanismo es el programático.

Soporte: Se dispone de un sitio para comunicarse y postear las dudas, además de un histórico de archivos, en el cual hay consultas realizadas por distintos usuarios con sus respectivas repuestas [25]. Se encuentra información desde el año 2000 a julio del 2009.

Integración con Eclipse: De [25] se obtiene una versión de Eclipse con los plugins necesarios para poder crear proyectos Repast.

Uso: Se dispone de dos mecanismos para la definición de los modelos. Uno es por medio de un editor gráfico donde se especifican diagramas de flujo, que definen el comportamiento de los agentes así como también las características del ambiente, mediante este mecanismo el framework genera automáticamente código Groovy. El problema que se presenta con esta representación es que la definición de modelos medianos o grandes con comportamientos complejos se vuelve sumamente engorrosa.
El segundo mecanismo es la implementación del modelo en Java, esta opción es mucho más viable para usuarios con experiencia en programación y conocimientos en dicho lenguaje.

Diseño del framework

Organización del código y estructura interna: El framework está compuesto por varios proyectos los cual evidencia un claro diseño modular. Las clases están debidamente comentadas y se cuenta con una completa API.

No se encontró un diagrama de arquitectura, pero investigando el código, se puede visualizar su estructura en paquetes, algunos de los más relevantes son:

- *Análisis*: Las clases en el paquete de análisis se utilizan para recopilar, registrar y graficar los datos.
- *Motor:* Estas clases son responsables de la creación, manipulación y simulación.
- *Evento*: Es responsable de la ejecución de las acciones de los agentes, como también de acciones en el propio modelo, como las actualizaciones de pantalla.
- *Juegos*: Contiene algunas clases para la ejecución de juegos de cooperación.
- *Gui*: Permite la visualización de la simulación, tiene la capacidad de tomar fotos instantáneas de la pantalla y hacer QuickTime de la visualización de películas a

medida que evoluciona con el tiempo. Además, hay algunas clases de utilidad que permite recoger algunas sencillas pero útiles estadísticas de la red.
- *Espacio*: Encontramos las relaciones entre los agentes.
- *Útil*: Contiene una variedad de clases de utilidad empleadas por RePast y por el modelador.

Modelado

Especificación del modelo: Como ya se ha mencionado existen dos formas de definir modelos, agentes y comportamientos.
Una es por medio de un entorno gráfico el cual provee una serie de bloques los cuales pueden ser bloques de inicio de método, bloque lógica, bloque *loop*, bloque *join*, bloque de decisión, etc.
Cuando se define un agente por este método se genera código automático en lenguaje Groovy.
La segunda forma es por medio de Java puro, se crean las clases que componen el modelo, en particular los agentes no deben de extender de alguna clase en particular solo se debe de definir el método que se ejecutará en cada paso de la simulación por medio de *annotations*.
Cuando se crea un proyecto Repast en Eclipse se crea automáticamente un archivo llamado model.score, este archivo es el que define la simulación. En este se especifica que agentes participarán de la simulación, cuál será su representación gráfica, como será la representación gráfica del ambiente, cuales son los parámetros que los usuarios podrán modificar... etc.

Formato de almacenamiento: La definición del modelo se organiza en dos niveles de almacenamiento, el primer nivel refiere a la implementación de las entidades que componen el modelo donde los formatos pueden ser clases Java o archivos con extensión .agent y .groovy.
El segundo nivel representa la definición de un escenario específico del modelo a ejecutar, este escenario se almacena en archivos .xml.

Entorno de simulación

Definición de una simulación: Una simulación se ejecuta a partir de la especificación del escenario, para acceder al entorno de simulación se debe ejecutar la clase principal *RepastMain* pasándole como atributo la ruta del directorio donde se encuentra el archivo model.score.

Manipulación de datos de salida: El entorno de simulación provee varios mecanismos que permiten analizar los datos resultados de la simulación. Se puede inspeccionar cada agente realizando doble clic sobre él, mostrando sus propiedades y valores respectivos. También permite realizar un Snapshot de la simulación (exportar a una imagen) y generar videos.

Avance en el tiempo: El avance del tiempo es del tipo time-step.

Tipo de inicialización: Desde el entorno de simulación se pueden editar los valores de los parámetros globales definidos en el modelo, también se puede definir sobre que atributos se desea generar gráficos así como especificar qué tipo de representación gráfica utilizar.

Uso de una semilla: Desde el entorno de simulación se permite definir o cambiar el valor de una semilla para la generación de una simulación. Por defecto genera la semilla a partir de la hora actual.

Controles de ejecución: El entorno de simulación provee cinco botones de control lo cuales permiten: inicializar, ejecutar, ejecutar paso a paso, parar y reiniciar la simulación.

Acceso a la simulación: Se provee funcionalidades para grabar una simulación en formato .xml, que permite volver a visualizarla. Permite también, visualizar una consola de logs, donde se puede observar si se han generado errores durante la simulación.

A. 4. Investigación – SeSAm (Shell for Simulated Agent Systems)

SeSAm [26] proporciona un entorno genérico para el modelado y la experimentación de simulaciones basadas en agentes. Sobre la base de un amplio número de componentes primitivos, el usuario es capaz de diseñar una simulación gráfica sin saber la sintaxis de un lenguaje de programación tradicional. La especificación del modelo es ejecutable en el mismo entorno, pudiéndose observar la dinámica de la simulación.

Criterios generales

Actualización y Mantenimiento: Es un framework mantenido, el lanzamiento de su última versión fue el 19 de enero del 2009 (release 2.5). A través de su página permite realizar reportes de bugs.

Documentación: La documentación es completa y organizada. Podemos encontrar:
Tutoriales para la creación de modelos, FAQ, tutoriales para desarrolladores (manuales para comprensión del mecanismo de plugins, ejemplos de código fuente, tutoriales de distribución (para trabajar con Eclipse), material adicional (slides, métricas de código, etc.).

Lenguaje de desarrollo/especificación: Java/-.

Licencia: LGPL.

Mecanismos de extensión: Para familiarizarse con extensiones de programación, se provee de un *Plugin Tutorial*.
Los plugins extienden las funcionalidades básicas de SeSAm. Pueden agregarse más primitivas de datos y modelos de comportamiento, que podrían ofrecer nuevas funciones en el menú principal, o bien añadir algunos editores. Los complementos se pueden agregar a SeSAm en una forma muy sencilla:

1. Descargando el Archivo-jar que contiene el plugin
2. Colocarlo en $SESAM_HOME$/plugins
3. Reiniciar SeSAm

Si bien SESAM ya ofrece muchas de las funciones primitivas y es muy poderoso, pueden crearse tipos de datos de usuario y funciones de usuario en la declaración de

un modelo, por lo que a menudo no es necesario programar nuevas funciones, ya que puede hacerse más fácil por la elaboración de modelos. Una ventaja es que el modelo se mantiene independiente de los plugins externos.

Soporte: Provee a los usuarios de una Wiki, en cuyo contenido se encuentra información general sobre el framework, documentación, herramientas de soporte de la comunidad y contacto con desarrolladores.

Integración con Eclipse: Puede utilizarse Eclipse como IDE, configurándose el CVS y descargando el proyecto SeSAm.
También puede generarse un .jar con las funcionalidades provistas por SeSAm y así distribuirse para utilizarse en algún otro proyecto como librería.
Puede integrarse Eclipse como librería, pero no es necesario hacer uso del IDE para el modelado y ejecución de una simulación, ya que provee un entorno propio.

Uso: Es un framework que provee presenta un entorno gráfico, si bien su uso parece bastante intuitivo, no resulta muy simple la especificación de los modelos. No es necesario tener un conocimiento de un lenguaje específico de programación para desarrollar una simulación, sin embargo se necesita tener conocimiento de la sintaxis de los lenguajes. La especificación del comportamiento va creando una estructura arborescente, ya que se realiza en notación postfija. Para realizar este tipo de construcciones es necesario que el usuario tenga conocimiento de la misma.

Diseño del framework

Organización del código y estructura interna: Todas las distribuciones contienen el código fuente (SeSAm_src.zip), aunque si se desea trabajar en un nuevo plugin se recomienda utilizar el CVS.
Inspeccionando el código fuente (el que se encuentra en la distribución), se puede observar que el código está organizado en diferentes directorios (paquetes) por área temática (aunque no se encuentra ningún documento que describa la organización de estos). Existe código que contiene comentarios descriptivos y código que no. En [26] se encuentra un tutorial (*SeSAm-Programming-Tutorial-Version3.doc*) que describe como programar nuevas funcionalidades para los agentes (brinda ejemplos de código).
La estructura interna no es modular y tampoco se compone de varias librerías, por el contrario todo el código se encuentra en una sola librería, la cual está organizada en diversos paquetes según la temática. Se provee de algunos diagramas (escasos y básicos), que grafican las dependencias entre algunas clases, pero la diagramación de la estructura es muy pobre.

Modelado

Especificación del modelo: El framework contiene un conjunto variado de modelos pre-armados entre ellos encontramos modelos de call centres, tráfico vial, presa-predador, propagación de información emergente, etc.

Las principales entidades de un modelo SeSAm son agentes, recursos y el mundo. Su estado y comportamiento pueden ser implementados sobre la base de programación visual (posee un editor gráfico). Para la especificación del estado no es necesaria la creación de ningún diagrama, se realiza a través del editor (al estilo Wizard).

Se utilizan diagramas de actividad basados en UML para especificar comportamiento, con algún agregado extra. Las actividades son consideradas como simples secuencias de comandos con inicio y fin sobre base de reglas. Las acciones y las condiciones pueden estar compuestas de un amplio número de bloques de construcción primitiva.

Concepto de agente en SeSAm: Las partes más importantes de los agentes son *reasoning engine* y *variables*. La primera es usada para especificar el comportamiento de los agentes y la segunda es para especificar el estado, basado en un conjunto de estados de variables.

Formato de almacenamiento: Permite almacenar el modelo generado (estado y comportamiento), en un archivo .xml.

Entorno de simulación

Definición de una simulación: La simulación puede ser ejecutada por diferentes situaciones y agregados en los llamados experimentos. También el modelo de instrumentación para la recolección de simulaciones y visualización de datos es posible a través de los llamados análisis. Antes de comenzar la ejecución de una simulación, el modelo se compila utilizando técnicas de optimización del compilador, por lo que el poder de la programación visual se combina con rápida ejecución.

Para la definición de una simulación es necesario definir un mundo (ambiente). Pueden definirse variables globales a este mundo. Y serán globales a la simulación (no de cada agente).

Elementos de simulación
Los elementos del modelo consisten en la descripción de los recursos, agentes, mundos, además de un conjunto de variables y comportamiento (una declaración sobre la base de determinadas normas y actividades). Para ejecutar una simulación, hay que definir algunos aspectos más:

- *Situations* - Son las configuraciones de un mundo dado con los ajustes de variables y algunos agentes que poseen una posición y configuración específica de manera que la ejecución de una simulación puede ser iniciada. Se puede ver como una situación(o estado) inicial.
- *Analisis List* - Es la definición de la posible instrumentación del modelo. Un análisis de lo que contiene información acerca de las selecciones de los datos (por ejemplo, el número de agentes) que se ha exportado a un medio (archivo, tabla, gráfico) y en qué condiciones (cada paso, en determinados intervalos, en determinadas situaciones).
- *Simulations* - Son combinaciones de una situación, un conjunto de elementos de análisis y una definición de una situación, cuando la simulación debe darse por concluida.
- *Experiments* - Son necesarios cuando la ejecución de la simulación debe ser automatizada. Sobre la base de acciones como el cambio de parámetros de una situación en la simulación y al ejecutarlo varias veces se puede generar una forma de escritura experimental, útil para la experimentación sistemática.

Para la especificación de la simulación se posee un editor gráfico, por lo que no es necesario el ingreso de código. Se puede inicializar los valores necesarios para llevarla

a cabo de una manera simple y visual. Por lo que resulta simple su definición, además podemos guardarla como .xml.

Manipulación de datos de salida: Para la definición de un gráfico se debe de utilizar el *Análisis.* Se encuentran diferentes seteos que pueden llevarse a cabo además de la edición de aspectos visuales.

Avance en el tiempo: El avance en el tiempo es del tipo time-step.

Tipo de inicialización: Existe inicialización del aspecto espacial de un agente (imagen, dirección, velocidad, posición). También existe inicialización de valores, estos pueden ser randomicos o personalizados (las distintas maneras de definirlos dependen del tipo de variable (dinámica o constante).

Uso de una semilla: No provee esta funcionalidad.

Controles de ejecución: Hay cuatro botones para controlar la ejecución de la simulación.
Play inicia la ejecución de una simulación, *Pause* detiene la ejecución, *Reset* restablece todos los valores y los objetos. La simulación puede ser reiniciada a mano. Un aspecto importante es el control *Step* (ejecución paso a paso), el cual es configurable con *Step Size* (cantidad de pasos).

Acceso a la simulación: Es posible almacenar los valores iniciales de una simulación (formato xml). No se provee funcionalidades para grabar una simulación y volver a repetirla (visualizar la misma ejecución previamente realizada). El framework permite visualizar una consola de logs, donde se puede observar lo ocurrido a lo largo de la ejecución de la simulación.

A. 5. Investigación – Ascape

Ascape [27] es un framework para el desarrollo y la exploración de propósito general basada en modelos de agentes. Ascape ofrece una amplia gama de herramientas de modelado y visualización.
Cuando se corre la simulación, es posible que personas no expertas, puedan modificar parámetros o reglas del propio modelo, como también ejecutar la simulación con las variedades que la misma brinda.

Criterios generales

Actualización y Mantenimiento: Es un framework mantenido, actualmente se encuentra en la versión 5.1.3 y su última actualización es del 20 de abril 2009.

Documentación: Su documentación es completa, contiene un manual de usuario, documentación técnica, un foro en línea y ejemplos de cómo usar Ascape integrado o no a Eclipse, según se elija.

Lenguaje de desarrollo/especificación: Java/Java.

Licencia: BSD.

Mecanismos de extensión: Ascape provee de una librería para su distribución y uso (jar), que si se desea extender se puede realizar modificando el código fuente, el cual se encuentra disponible en [27].

Soporte: En caso que se quiera contribuir con el código de Ascape, se especifica una dirección de correo con la cual nos podemos poner en contacto con el desarrollador, también posee un foro en línea al cual se puede acceder.

Integración con Eclipse: Se integra fácilmente con Eclipse, en la documentación provista por la distribución se detallan los pasos a seguir.
Es posible acceder al código fuente que se encuentra en el repositorio del sitio oficial [27].

Uso: Para la creación de modelos basados en agentes, es necesario contar con usuarios que tengan conocimientos de programación (específicamente Java). Se tienen las clases necesarias para implementar agentes y reglas de comportamiento. Con respecto a la simulación y uso del framework para el usuario final, no requiere que los usuarios tengan conocimiento en programación, dado que la interfaz que brinda es bastante intuitiva, permite modificar parámetros y reglas desde el propio modelo. Con respecto a la ejecución de la simulación y salidas, también es bastante sencillo, dado que el framework cuenta con botones que le permite al usuario indicar, el tiempo que quiere correr la simulación, además permite rápidamente indicar si quiere o no gráficas, estadísticas e indicar el formato de las mismas.

Diseño del framework

Organización del código y estructura interna: Se puede observar que el código está organizado en distintos directorios según área temática. El diseño es en capas, con el código fuente documentado, comprensible y "prolijo".
Ascape está desarrollado con programación modular, no encontrándose ningún diagrama que especifique más detalladamente la estructura interna que lo compone.

Modelado

Especificación del modelo:

- *Construcción de Agentes Ascape.*
  Hay tres propiedades básicas que deben ser asignadas:
  1) El tipo de agentes que contiene
  2) El orden de ejecución de los agentes
  3) (opcionalmente) Las operaciones estadísticas a realizar sobre los agentes

- *Especificación de los Agentes en el Modelo.*
  Los agentes son creados a partir de clases java, donde también se describen sus propiedades, comportamientos y reglas.
  Para crear un modelo Ascape, se necesitan dos archivos de Java:
  1) Una clase que define el propio modelo.
  2) Una clase especificando algún tipo de agente y sus reglas de comportamiento.

- *Reglas en Ascape.*
  Las reglas no están vinculadas a los agentes individuales, sino al ambiente en los cuales se encuentran.
  Intuitivamente, cada agente tendrá el mismo conjunto de reglas.
  Una ventaja de tener las reglas para los agentes en el propio modelo, es que se da al usuario la posibilidad de cambiar las reglas en tiempo de ejecución. En las propiedades del modelo se puede editar la configuración, mientras que la simulación se esté ejecutando. Esto permite al usuario cambiar el orden de las normas, sincrónica o asincrónica, y permite eliminar algunas reglas de la conducta del agente, sin tener que cambiar el código.

Formato de almacenamiento: El modelo se guarda como clases Java.

Entorno de simulación

Definición de una simulación: Permite definir tanto simulación como también estadísticas.
Con respecto a los valores de inicialización o parámetros para la simulación, estos se pueden editar desde la ventana de configuración, en tiempo de ejecución si así se quisiera.

Manipulación de datos de salida: Ascape permite grabar la simulación en una película QuickTime. Presentando como limitación que debido a cambios en QuickTime, puede pasar que las películas no funcionen correctamente en ediciones de Mac OS X.

Avance en el tiempo: El avance del tiempo es del tipo time-step.

Tipo de inicialización: La inicialización se puede dar en la propia clase Java cuando se ejecuta o en la ejecución del modelo, haciendo Stop, modificando los parámetros y volviendo a retomar la ejecución.

Uso de una semilla: No se encontró información al respecto.

Controles de ejecución: Se puede tener control de la simulación, de las siguientes maneras:
1. Ir al principio de la simulación
2. Reiniciar la simulación
3. Parar la simulación
4. Pausar la simulación
5. Realizar una iteración (1 Step), mientras el actual modelo se encuentra en un estado detenido.

Acceso a la simulación: Se puede grabar la simulación pero no se garantiza que funcione, depende de la plataforma en la cual corre (ver limitaciones). De todas maneras es posible guardar el modelo a partir de determinado tiempo de ejecución para luego retomarlo.

Apéndice B
Listado completo de frameworks que conformaron el estudio.

Se detalla el estudio realizado a los veintidós frameworks de simulación basados en SMA, que se encontraron de una búsqueda exhaustiva en Internet y textos recomendados, se analizaron con el objetivo de encontrar el que más se adecue a las necesidades planteadas en el proyecto.
El primer paso fue explorar e investigar en Internet los distintos frameworks y a medida que la investigación fue avanzando surgiendo nuevos framework a investigar los cuales fueron agregados a la evaluación.

B.1. AgentSheets

Descripción: [35] Es un framework basado en la simulación de agentes que permite a una amplia gama de usuarios finales (desde los niños hasta los profesionales) crear sus propias simulaciones interactivas y juegos. AgentSheets combina agentes, hojas de cálculo y tecnologías de autoría en Java en un sólo medio. Está basado en la programación táctil, extendiéndose más allá de la programación visual.

Actualización y Mantenimiento: Si bien no se detalla la fecha de la última liberación, visitando el sitio web (actualizado al 2009) se observa su continuo desarrollo, mantenimiento y uso.

Documentación: Es muy completa y comprende: manuales de usuario, videos al estilo paso a paso (guían en la construcción de modelos), FAQ, libros recomendados.

Lenguaje de programación: Java.

Licencia: Es comercial, libremente solo se encuentran versiones trial.

Entorno de simulación: Provee un editor de propiedades de simulación, el usuario puede inspeccionar y editar las propiedades, además de personalizar su representación gráfica.

Mecanismos de extensión: No lo permite.

Modelado: Permite la representación de agentes, brinda un editor para poder definir entidades, además de poderle asociar gráficos representativos. Por medio del editor se puede especificar el comportamiento, basado en una estructura muy simple de sentencias *if-then*. Es un framework muy intuitivo que permite la creación de modelos rápidamente y sin necesidad de tener conocimientos en programación.

B.2. AgentTools

Descripción: [36] Está diseñado para ayudar a los usuarios a analizar, diseñar e implementar sistemas Multi-Agentes por medio de una programación gráfica.
El entorno de diseño de la aplicación, define un comportamiento de alto nivel gráfico, usando la metodología MaSE (Multiagent System Engineering).

Actualización y Mantenimiento: La última modificación fue en setiembre de 2008 y su versión es 1.0.9.

Documentación: Cuenta con una documentación básica, tiene demos que explican el paso a paso para la creación de modelos y documentación on-line.

Lenguaje de programación: Java.

Licencia: GNU/GPL.

Entorno de simulación: No provee un entorno de simulación.

Mecanismos de extensión: Cuenta con plugins que se pueden agregar y actualizar.

Modelado: Permite modelar agentes, sus relaciones y reglas, como también el ambiente en el cual se encuentran. Se programa todo usando una notación gráfica AOSE (Ingeniería de Software orientada a Agentes).

B.3. Breve

Descripción: [37] Es un paquete de software que facilita la construcción de simulaciones en 3D de los sistemas Multi-Agente y la vida artificial. Permite definir los comportamientos de los agentes en un mundo (ambiente) y observar cómo interactúan. Breve incluye la simulación física y la detección de colisiones de manera que puede realizar simulaciones más realistas, y provee un motor de visualización de OpenGL que permite visualizar el mundo simulado.

Actualización y Mantenimiento: La última versión disponible es del 19 de febrero del 2008. No es un framework en desuso y se está extendiendo e implementando nuevas versiones.

Documentación: Posee una básica documentación on-line, reportes de bugs y un foro para usuarios. Además se encuentra documentada una API que posee las clases necesarias para la creación de una simulación.

Lenguaje de programación: Utiliza un lenguaje de scripting llamado "Steve" o Python. Python surge en las últimas liberaciones de este software, por tanto no es tan robusto como Steve.

Licencia: GPL.

Entorno de simulación: Provee un marco de simulación 3D, basado en simulaciones de vida artificial y sistemas descentralizados.

Mecanismos de extensión: Presenta una extensible arquitectura de plugins que le permite escribir sus propios plugins e interactuar con su propio código. La escritura de plugins permite ampliar Breve para trabajar con los proyectos existentes.

Modelado: Breve es un modelo de agentes basados en el medio ambiente en donde los comportamientos de los agentes se simulan en cada paso de tiempo. Para escribir una simulación en Breve, deben definirse agentes en términos de cómo se comportan y cómo interactúan. Estos comportamientos se actualizan en cada paso de tiempo de manera que los Agentes están continuamente reaccionando en su entorno. En cada paso de tiempo, el Agente utiliza su método *iterar* para llevar a cabo una serie de tareas: obtener información sobre el mundo a través de sensores, realizar el cómputo de la información y, finalmente, cambiar su comportamiento o el estado sobre la base de la información computarizada.

B.4. Cormas

Descripción: [03] Es un framework para simulaciones Multi-Agente, enfocado hacia la administración de recursos naturales renovables.

Actualización y Mantenimiento: La última versión disponible es de marzo de 2008.

Documentación: Es escasa, se encuentran tutoriales de Smalltalk y varios ejemplos de uso del framework. Además se proporcionan los diagramas de clases como referencias para el desarrollo.

Lenguaje de programación: Es una aplicación basada en VisualWorks, es un entorno de programación que permite el desarrollo en lenguaje O.O. Smalltalk.

Licencia: LGPL.

Entorno de simulación: Provee un entorno de simulación, simple e intuitivo. Para poder ejecutar una simulación inicialmente se debe definir un escenario. Permite editar determinadas propiedades de la simulación (personalizar valores, definirlos randomicos, etc.), también se puede setear la cantidad de pasos de ejecución, a nivel grafico se puede escoger el tipo de visualización deseada (*probes, Messages, Space*).

Mecanismos de extensión: Se puede extender la herramienta desarrollando código Smalltalk.

Modelado: Se pueden representar agentes, recursos y ambiente, mediante el entorno gráfico (a modo de wizard). Para la especificación del comportamiento se debe programar usando el lenguaje Smalltalk, lo cual no resulta muy práctico, ya que se necesita poseer conocimientos sobre el mismo.

B.5. ECHO

Descripción: [38] Es un framework de simulación desarrollado para la investigación de mecanismos que regulan la diversidad y el intercambio de información en los sistemas de procesamiento, compuesto de múltiples agentes que interactúan en complejos sistemas adaptativos (CAS). Genotipos individuales codifican las normas de las interacciones. En una típica simulación, las poblaciones de estos genomas evolucionan en redes de interacción que regulan el flujo de recursos. Se asemejan a las redes de comunidades de especies en los sistemas ecológicos. Echo define los parámetros de flexibilidad y las condiciones iniciales para que los investigadores puedan llevar a cabo una serie de experimentos.

Actualización y Mantenimiento: La última versión disponible es del 19 de febrero de 2002. La versión actual es 1.3 beta 2, pero la interfaz gráfica de usuario no está disponible. Es una herramienta en desuso y obsoleta.

Documentación: Es incompleta, la mayoría de los links que se ponen como referencia no se encuentran disponibles. Hay un mensaje en la página diciendo:
"*Por favor no nos envíe correo preguntando cómo compilar versiones anteriores de echo. Los widgets no compilan y en versiones anteriores no están soportados.*"
Sin embargo, hay un documento que describe cómo utilizar esta aplicación, detallando cómo compilar e instalar el código fuente en C.

Lenguaje de programación: El código fuente se encuentra en C, pero se menciona la existencia de una versión en Java.

Licencia: Es de libre distribución, pero el link de descarga no se encuentra disponible.

Entorno de simulación: No se tuvo acceso a la documentación, para poder detallar este ítem.

Mecanismos de extensión: No se tuvo acceso a la documentación, para poder detallar este ítem.

Modelado: No se tuvo acceso a la documentación, para poder detallar este ítem.

B.6. Ingenias

Descripción: [39] Ingenias Development Kit (IDK), es un conjunto de herramientas para la especificación, verificación y realización de aplicaciones basadas en sistemas Multi-Agentes. Se orienta a investigadores que buscan trabajar con herramientas orientadas a la especificación de agentes, ya que es una plataforma para el modelado de este tipo de sistemas.

Actualización y Mantenimiento: La última versión disponible es del 30 de abril de 2008.

Documentación: Es completa y organizada, podemos encontrar una guía para la instalación, manejo gráfico, y generación de módulos, entre otros.

Lenguaje de programación: Java.

Licencia: GPL.

Entorno de simulación: No posee, es sólo un framework para modelado.

Mecanismos de extensión: Sus funcionalidades pueden ampliarse con módulos que realizan generación automática de código y verificación de las especificaciones. Algunos de estos módulos se ofrecen con la distribución de IDK, pero el desarrollador también puede crear nuevos módulos para una aplicación. Esta característica cuenta con el apoyo para acceder a la IDK MARCO (un API para módulos de programación y generación de herramientas de módulos).

Modelado: IDK puede ser directamente utilizada para especificar SMA, utilizando un editor gráfico. El objetivo principal del editor es crear y modificar las especificaciones de sistemas Multi-Agentes. Una especificación es un conjunto de diagramas que representan diferentes puntos de vista de un SMA. Los diagramas constituyen un proyecto, y se organizan utilizando paquetes como construcciones. El editor guarda estas especificaciones en formato XML, de modo que otras herramientas externas puedan analizarlas y producir otro tipo de productos. Además, el editor ofrece acceso en tiempo de ejecución para instalar plugins, y es capaz de cargar nuevos plugins en tiempo de ejecución.

B.7. Jack

Descripción: [40] Jack fue desarrollado para proveer una extensión de Java orientada a agentes.
Según se establece, las metas más importantes en su diseño fueron proveer a los desarrolladores de un producto robusto, estable y "liviano", satisfacer una variedad de necesidades prácticas, facilitar la transferencia de tecnología de la investigación a la industria, y permitir la investigación aplicada.

Actualización y Mantenimiento: Última versión disponible es la 5.4.

Documentación: Es completa y organizada, se encuentran tutoriales, FAQs (programadores, para instalación y para profundizar en conceptos de agentes).

Lenguaje de programación: Provee un entorno de desarrollo orientado a agentes construido sobre Java y completamente integrado con este lenguaje de programación. Incluye a Jack Agent, un lenguaje orientado a agentes y utilizado para implementar sistemas de software orientados a agentes. Jack Agent no solo extiende la funcionalidad de Java, sino que además provee un entorno para soportar un nuevo paradigma de programación. El código Jack es primero compilado a código Java antes de ser ejecutado. La relación entre Jack y Java es análoga a la relación entre los lenguajes C++ y C.

Licencia: Registrándose en la página se obtiene una versión por 60 días.

Entorno de simulación: Para la simulación Jack ofrece *relojes* de tipo real, en el cual sincroniza los agentes con el tiempo real, como también *relojes* del tipo simulado, en el cual el tiempo puede ser marcado manualmente para proporcionar más control sobre el paso del tiempo, en un entorno de simulación.
Ofrece para la simulación gráfica: plan de seguimiento, diseño de diagrama de localización y diagrama de interacción de agentes.

Mecanismos de extensión: Todas las formas en las que extiende a Java, son implementadas como plugins, lo que permite que el lenguaje sea lo más extensible y flexible posible.
Desde la óptica de un programador, Jack ofrece tres extensiones a Java: un lenguaje de agentes, un compilador de agentes y un kernel de agentes.

Modelado: Jack extiende en tres niveles:

1. Definición: Ofrece un conjunto de clases, orientada a la construcción de los agentes permitiendo declarar agentes, eventos y planes.
2. Declaración: Este nivel ofrece un conjunto de declaraciones, que permiten identificar relaciones entre las clases mencionadas en el nivel anterior.
3. Instrucción: Provee un conjunto de declaraciones que permiten operar con las estructuras de datos de los agentes declarados.

B.8. JADE

Descripción: [41] Java Agent Development Framework (JADE), simplifica la implementación de SMA a través de un middleware que cumple con las especificaciones FIPA [42] y a través de un conjunto de herramientas gráficas que soporta la depuración y despliegue. El framework, permite que los agentes puedan ser distribuidos a través de máquinas (sin necesidad de compartir el mismo sistema operativo) y la configuración puede ser controlada a través de una interfaz gráfica de usuario remotamente. La configuración se puede cambiar incluso en tiempo de ejecución por los agentes en movimiento de una máquina a otra, cómo y cuando sea necesario.

Actualización y Mantenimiento: La última versión disponible es del 4 de abril del 2008. Es una herramienta muy usada y mantenida.

Documentación: Existe una completa documentación disponible, cuenta con una comunidad, news, guías para usuarios, guías para programadores, ejemplos y una completa API.

Lenguaje de programación: Java.

Licencia: LGPL.

Entorno de simulación: Permite la realización y definición de simulaciones, pero basada en la observación de las interacciones de los agentes (involucrados) vía mensajes. No cuenta con un motor de simulación con las características requeridas, pero provee todos los mecanismos para el desarrollo del mismo.

Mecanismos de extensión: Permite desarrollar nuevos módulos (debiéndose extender de algunas clases básicas). Proporciona una API para creación de agentes y elementos

relacionados. La dificultad de esta plataforma puede encontrarse en su generalidad, ya que es muy amplia.

Modelado: Para generar un modelo de debe desarrollar una serie de archivos Java. Se deberá definir una clase diferente por cada tipo de agente (debiendo extender de la clase *Agent*) y para cada comportamiento definido sobre el agente se debe de crear una clase (debiendo extender de la clase *Behaviour*). Además se pueden crear otros archivos Java para definir la ontología empleada por los agentes.

B.9. MadKit

Descripción: [43] Es un framework Multi-Agente para desarrollar y ejecutar aplicaciones basadas en un enfoque orientado a la organización. Este enfoque utiliza a los grupos y los roles como base para construir aplicaciones complejas. MadKit no está asociado a ninguna arquitectura de agentes en particular, permitiendo a los usuarios de esta plataforma implementar libremente sus propias arquitecturas.
La comunicación en MadKit se basa en un mecanismo de peer to peer, y permite a los desarrolladores a la hora de desarrollar aplicaciones distribuidas utilizar principios Multi-Agentes.

Actualización y Mantenimiento: La última versión disponible es del 29 de noviembre de 2005. Su último release es el 4.1.2.

Documentación: Es básica y bien organizada, brinda manuales al estilo paso a paso (en francés actualizada 2009), una descripción de la API (a nivel del kernel), un foro (no muy usado, solo 24 posts), FAQ. Si bien la documentación es pobre, se puede deber a que la construcción del sitio oficial es bastante reciente (2008), el cual está siendo actualizando.

Lenguaje de programación: Soporta los siguientes lenguajes Java, Jess, Python, Scheme, BeanShell. Permitiendo que otro lenguaje de script puede ser fácilmente añadido.

Licencia: GPL/LGPL.

Entorno de simulación: No permite realizar simulaciones. Lo única manera de interacción entre agentes que se muestra es la comunicación pero a nivel del envío de mensajes. Necesita interoperar con otros sistemas (Warbot y TurtleKit) para contar con un motor de simulación.

- *TurtleKit* - Es un framework para la construcción basada en Logo de simulaciones Multi-Agentes con Madkit.. A diferencia de StarLogo o NetLogo, TurtleKit propone un enfoque de programación de Logotipo dando todas las posibilidades que ofrece el alto nivel de los lenguajes de programación como Java y Python.
- *Warbot* - Se orienta a "la guerra de robots", o sea simulaciones de dos equipos de robos enfrentados en una competencia.

Mecanismos de extensión: No se encontró información al respecto.

Modelado: MadKit está construido bajo el concepto de Agente/Grupo/Rol.
Un agente es especificado como una entidad activa que se comunica, la cual posee roles dentro de los grupos. Esta definición de agentes es intencionalmente general para permitir a los diseñadores adoptar el modelo de agente más preciso relacionado con sus aplicaciones.
Los grupos son definidos como conjuntos atómicos de agregaciones de gentes, o sea, cada agente es parte de uno o más grupos.
El rol es una representación abstracta de la función de un agente, servicio o identificación dentro de un grupo.

B.10. Magsy

Descripción: [44] Es un framework para el desarrollo de aplicaciones Multi-Agentes. No se encontró material sobre esta herramienta, solo se puede describir las propiedades que se mencionan a continuación.

Actualización y Mantenimiento: La última versión disponible es del año 1993. Su último release es 3. Es un framework que se encuentra obsoleto y en desuso.

Documentación: Presenta una escasa documentación, no se encuentra manual de usuario, ni manual técnico.

Lenguaje de programación: El lenguaje es OPS5.

Licencia: Es libre de código cerrado.

Entorno de simulación: Provee entorno de simulación, pero no se detalla información al respecto.

Mecanismos de extensión: No se encontró en la documentación, información al respecto.

Modelado: Permite modelar agentes, los cuales cada uno de ellos tiene un intérprete de reglas

B.11. MAML

Descripción: [45] Es una extensión del lenguaje SWARM desarrollado por estudiantes, el objetivo del framework es crear modelos basados en agentes con poca programación, de todas maneras es un proyecto en construcción (aun no está terminado).

Actualización y Mantenimiento: Se discontinuo y se encuentra en una fase pre-beta, su última modificación fue en el año 2000 encontrándose en la versión 0.03.2 Alfa.

Documentación: Cuenta con una escasa documentación, tiene un manual técnico y algunos ejemplos.

Lenguaje de programación: Se programa en lenguaje Objetive C.

Licencia: GNU.

Entorno de simulación: El desarrollo de MAML, fue dedicado a simplificar el esfuerzo de programación en lugar de proporcionar un entorno para la simulación, de todas maneras permite interoperar con Swarm.

Mecanismos de extensión: Permite desarrollando código fuente.

Modelado: Permite modelar agentes, variables y subrutinas.
Los agentes son definidos en clases, los cuales heredan de alguna clase agente definida por el usuario o por la clase SwarmObject (clase definida en la librería Swarm). El modelo se define en una clase que hereda de GUISwarm, el cual brinda funcionalidades para el desarrollo gráfico.
Cabe destacar que MAML no cuenta con un entorno de desarrollo, es necesario escribir el programa en archivos de texto con las extensiones correspondientes.

B.12. MIMOSE

Descripción: [46] El sistema de modelado y simulación MIMOSE consiste en un lenguaje de descripción de modelos y un marco experimental para la simulación. El objetivo principal del proyecto MIMOSE fue el desarrollo de un lenguaje de modelado que considera las demandas especiales de los modelos en las ciencias sociales, especialmente la descripción de las relaciones no lineales, cuantitativas y cualitativas, las influencias estocásticas y procesos de nacimiento y muerte.

Actualización y Mantenimiento: Ultima actualización año 2000, la versión release para descargar es la 2.0

Documentación: Básica (manual de usuario).

Lenguaje de programación: Java, pero para la especificación de modelos se utiliza un dialecto propio.

Licencia: Libre (código cerrado).

Entorno de simulación: Provee un entorno de simulación que permite manipular los valores iniciales. La salida la realiza por medio de gráficos.

Mecanismos de extensión: No permite.

Modelado: La definición del modelo se realiza por medio del lenguaje ofrecido por la herramienta. La codificación del modelo se puede realizar desde el entorno grafico provisto por MIMOSE o cargando un archivo plano.

B.13. NetLogo

Descripción: [47] Es un entorno de modelado programable para simular los fenómenos naturales y sociales. NetLogo es especialmente adecuado para el modelado de sistemas complejos que se desarrollan en el tiempo. Es también un entorno de edición que permite a los estudiantes, profesores y desarrolladores crear sus propios modelos, debido a la simplicidad que proporciona la plataforma. Además, es lo suficientemente avanzado para servir como un poderoso framework a los investigadores en muchos campos.

Actualización y Mantenimiento: La última versión disponible es del 10 de junio de 2009. Su último release es NetLogo 4.1RC2, es un framework en continuo desarrollo y muy usado.

Documentación: Posee una completa documentación, en la que podemos encontrar: lista de bugs, FAQ, guías de usuario, una comunidad.

Lenguaje de programación: Es una plataforma desarrollada en Java, incluye su propio lenguaje (Logo) de programación para la especificación del modelado, se hace referencia en su facilidad para el uso.

Licencia: Si bien el código no es abierto se provee de una API la cual puede ser extendida.

Entorno de simulación: Posee un gran número de funcionalidades, muchas de las acciones de la simulación se definen con el dialecto propietario, por lo visto en varios ejemplos, se pueden asignar valores randómicos de inicialización a los agentes, definir el tamaño de la muestra, asignar colores y vistas a los agentes y ambiente, definir diferentes posiciones y accionar (comportamiento). Se puede con los datos obtenidos exportar imágenes, realizar gráficos.

Mecanismos de extensión: Tiene todo el código en un archivo. No permite un esquema organizacional como Java (esto puede traer complejidad para modelos grandes).
No permite acceso inmediato a los algoritmos que implementan las primitivas de reproducibilidad.
NetLogo contiene "Biblioteca de Modelos", que es una gran colección de simulaciones pre-escritas que se pueden utilizar y modificar. Estas simulaciones abordan muchas áreas de contenido en las ciencias naturales y sociales, incluyendo la biología y la medicina, la física y la química, las matemáticas y la informática, la economía y la psicología social.

Modelado: Los modelos se pueden ejecutar como applets de Java dentro de un navegador web.La biblioteca de modelos antes descripta, se compone de cinco categorías: Sample Models, Perspective Demos, Curricular Models, Code Examples, y HubNet Computer Activities. Según lo investigado si se desea construir un modelo, se debe de utilizar uno provisto por la biblioteca, o por el contrario se deberá desarrollar desde código un prototipo de modelo desde el inicio.

B.14. PS-I

Descripción: [48] Es un entorno para ejecutar simulaciones basadas en agentes.
Fue diseñado con el objetivo de ser una aplicación sencilla de usar por usuarios que no tienen conocimiento en programación.

Actualización y Mantenimiento: La última actualización se realizo en el año 2005, la versión es 3.0.6.

Documentación: Contiene una documentación muy escasa, solo se encontró una documentación muy básica de definiciones que usa el framework.

Lenguaje de programación: La interfaz de usuario ha sido implementada con TCL/TK, lo cual permite que todo el código sea compartido tanto en Linux como Windows. TCL (Tool Command Language) es un lenguaje de Scripts. La combinación de Tcl con Tk (Tool Kit) es conocida como Tcl/Tk, y se utiliza para la creación de interfaces gráficas.

Licencia: GPL.

Entorno de simulación: Permite ver la simulación avanzando con tiempo TimeStep, también permite visualizar estadísticas que permite exportar dicha salida a CSV (Comma Separated Values), un tipo de archivo que se utiliza para importar datos desde una aplicación de software a otro, con comas que separan los valores en cada campo.

Mecanismos de extensión: No se encontró información en la documentación.

Modelado: No necesita programación, permite definir agentes, reglas y declarar los parámetros, todo por medio de ventanas, donde se va completando la información necesaria según corresponda.

B.15. SimAgent

Descripción: [49] Está diseñado principalmente para apoyar el diseño y aplicaciones de agentes complejos. Originalmente fue desarrollado para apoyar la investigación exploratoria sobre el hombre como agente inteligente, pero también ha sido utilizado para proyectos de estudiantes en el desarrollo de una variedad de juegos interactivos y simulaciones.

Actualización y Mantenimiento: La última versión disponible es del 30 de mayo de 2005.

Documentación: Es básica, se encuentra un tutorial, varios documentos y publicaciones.

Lenguaje de programación: Se encuentra escrito en Pop-11, el cual es muy similar en el poder y la diversidad de Common Lisp, pero con una sintaxis más convencional (entre algunas de sus diferencias). Por ejemplo, no tiene intérprete, sólo un compilador incremental.

Licencia: Open Source.

Entorno de simulación: Posee un entorno de simulación, pero la documentación investigada no detallaba ni especificaba sus potencialidades.

Mecanismos de extensión: SimAgent hace uso de las instalaciones orientadas a objetos en el paquete Pop-11 *objectclass*, como una extensión CLOS-Pop-11 para proporcionar la herencia múltiple, funciones genéricas, etc. Pop-11 se convierte en un lenguaje para el diseño reutilizable y extensible de módulos de software, como se hace para CLOS Lisp.

Modelado: Permite definir agentes que interactúan entre si y características del ambiente en donde se encuentran los agentes.

B.16. SimPack

Descripción: [50] Es un framework que proporciona a los usuarios un punto de partida para la simulación de un determinado sistema. SimPack se destina principalmente para investigar la similitud entre varios conceptos de ontologías y entre varios códigos fuente.

Actualización y Mantenimiento: La última versión disponible corresponde al año 2006.

Documentación: Proporciona una completa documentación en donde encontramos un manual técnico, información acerca de las versiones libreadas y una lista de correo a la cual se puede suscribir.

Lenguaje de programación: Originalmente fue escrito en lenguaje C, el cual ya no se mantiene, la nueva versión está escrita en Java.

Licencia: GPL.

Entorno de simulación: Provee entorno de simulación, admite una amplia variedad de eventos y la programación de modelos de simulación se hace en tiempo continuo.

Mecanismos de extensión: No se encontró información al respecto.

Modelado: No es framework que permita modelado, solo provee un entorno de simulación.

B.17. StarLogo

Descripción: [51] Es una extensión del lenguaje Logo, adecuado para el modelado de sistemas complejos y descentralizados.

Actualización y Mantenimiento: Última actualización julio 2008, se encuentra en versión 2.2.

Documentación: Presenta una documentación completa, cuenta con lista de errores, lista de correos, FAQ y manuales on-line que indican los pasos para crear un proyecto en StarLogo.

Lenguaje de programación: StarLogo (extensión de Logo).

Licencia: La licencia es de uso libre y su distribución es para usos no comerciales, el código es cerrado.

Entorno de simulación: Permite visualizar la simulación como también obtener gráficos e histogramas.

Mecanismos de extensión: No se encontró información al respecto.

Modelado: Permite definir *tortugas*, *parches* y *observadores*.
Tortugas: Representa a cualquier tipo de objetos (por ejemplo, un coche, una hormiga, una molécula, etc.), cada una de ellas tendrá una posición y un color, teniendo la posibilidad de editar sus parámetros, así como también definir funcionalidades. Permite controlar las acciones e interacciones de varias tortugas en paralelo.
Parches: Objetos pasivos que pueden ejecutar comandos StarLogo y pueden actuar sobre las tortugas o parches.
Observadores: Son responsables de mirar a las tortugas y parches, permitiendo crear nuevas tortugas y supervisar la actividad de las tortugas y parches existentes.

B.18. SugarScape

Descripción: [52] Es un framework enfocado en la realización de estudios de fenómenos sociales humanos, incluido el comercio, la migración, la formación de grupos, la interacción con un medio ambiente, la transmisión de la cultura, la propagación de enfermedades, y la población dinámica.

Actualización y Mantenimiento: La última versión disponible es del 27 de agosto de 2005. Su último release es el 2.44.

Documentación: Es escasa y desorganizada, la descarga del framework contiene documentación, además del acceso a un news, listas de bugs.

Lenguaje de programación: Java.

Licencia: GPL.

Entorno de simulación: Posee un entorno de simulación, pero la documentación investigada no detallaba ni especificaba sus potencialidades.
En el sitio web del framework se encuentra un applet, que permite ejecutar un ejemplo de simulación, pero no es algo muy descriptivo ni intuitivo.

Mecanismos de extensión: Permite extenderse, desarrollando su código fuente.

Modelado: En SugarScape se puede representar el ambiente compuesto de células en una estructura 2D, cuenta con las siguientes clases:

- *Cell*: Corresponde a las células.

- *CellSpace*: Define el ambiente, implementa los métodos del mismo, contiene a las células que ocupan cada casillero de la red.
- *Citizen*: Permite representar al ser humano, el cual posee atributos, vive en un periodo de vida.
- *GoLconst*: Permite configurar y manipular el comportamiento y propiedades de los objetos que participan en la simulación.

B.19. Swarm

Descripción: [53] Es un framework de propósito general para simular sistemas complejos adaptativos.
El componente principal que organiza los agentes es un Swarm, es una colección de agentes con su scheduller de actividades. Los agentes interactúan unos con otros a través de eventos discretos.
Podemos encontrar Swarm (Objetive-C) y Java Swarm. La primera son librerías escritas en Objetive-C y la segunda es un conjunto de clases Java que permiten el uso de las librerías de objetos desde Java.

Java Swarm
Cumple con el objetivo de llevar Swarm a usuarios Java. En un principio era un buen entorno ya que no existían muchos otros en Java, pero luego que comenzaron a surgir otros que evidenciaron los errores del framework.

Objetive-C Swarm
Es estable, tiene pocos bugs. Es pequeño y bien organizado, además provee un set de herramientas completas. Tiene una base conceptual clara y un diseño inteligente. Permite la separación de las interfaces gráficas y el modelo.

Actualización y Mantenimiento: La última versión disponible es de febrero de 2005. Su último release es el 2.2.

Documentación: Es completa, se encuentran guías de usuario, manuales, Api y aplicaciones. No se tiene ningún contacto personal con el Grupo de Desarrollo de Swarm. El soporte está disponible a través de listas de correo y preguntas frecuentes.

Lenguaje de programación: El kernel está desarrollado en Objetive-C, pero existen librerías para uso del kernel desarrolladas en Java.

Licencia: GPL.

Entorno de simulación: No provee un entorno gráfico para la especificación de una simulación, debe desarrollarse en código lo necesario.

Mecanismos de extensión: Permite extenderse, a través de módulos generados a partir del kernel (Por Ej. Java Swarm).

Modelado: Swarm ofrece aplicaciones que ilustran el uso del framework. Hay, sin embargo, plantillas ya preparadas para la especificación de modelos que pueden utilizarse para el área de ciencias sociales. No se provee una interfaz de usuario para el modelado, pero permite el desarrollo de una. Es necesario tener conocimientos de programación para el uso Swarm.

B.20. TeamBots

Descripción: [54] TeamBots es un framework para la simulación de SMA para la investigación de robótica móvil. Apoya prototipos, simulación y ejecución de sistemas de control multi-robot.

Actualización y Mantenimiento: La última versión disponible es del 14 de abril de 2000.

Documentación: Es básica y organizada. Actualmente TeamBots está documentada principalmente utilizando javadoc (se encuentran enlaces). Una de las metas futuras es la creación de un completo tutorial para usuarios.

Lenguaje de programación: Java.

Licencia: Puede utilizarse libremente para la educación y la investigación. El uso comercial está permitido siempre que el usuario no se beneficie del software TeamBots. Otros usos no están permitidos sin el permiso escrito.

Entorno de simulación: Una de las características más importantes del medio ambiente TeamBots es que se apoya en la simulación de prototipos de los mismos sistemas de control que se pueden ejecutar en robots móviles.
El entorno de simulación TeamBots es extremadamente flexible. Soporta múltiples robots con hardware heterogéneos y funcionamiento con heterogéneos sistemas de control. Complejos (o simples), entornos de experimentación se puede diseñar, incluyendo determinados objetos en la simulación a través de una fácil edición.

Mecanismos de extensión: No se encontró información al respecto.

Modelado: No se encontró información al respecto.

B.21. VSEit

Descripción: [55] Fue desarrollado para apoyar la construcción de modelos en la economía y las ciencias sociales, en la investigación y en enseñanza. Algunos de estos modelos, así como la dinámica de los sistemas de autómatas celulares fueron utilizados para el dictado de cursos de ciencias sociales y ciencias del medio ambiente de simulación.

Actualización y Mantenimiento: Su último release es el 2.0, pero no se pudo determinar la fecha del mismo.

Documentación: Es básica, se encuentran guías de usuario, guía para el modelado y ejemplos de simulación.

Lenguaje de programación: Java.

Licencia: Para recibir los archivos VSEit que contienen las clases Java hay que ponerse en contacto con el desarrollador vía correo electrónico. No se encontró un link de descarga, el código fuente es propiedad del desarrollador.

Entorno de simulación: Las funciones de control de la simulación y registro son bastante simples. Los programas de simulación VSEit se proporcionan como applets.
Cada simulación se ejecuta con un conjunto específico de parámetros del modelo y valores iniciales, así como los ajustes para ver los resultados de la simulación, permitiéndose ejecutar varias simulaciones simultáneamente.

Mecanismos de extensión: Está orientado a facilitar la incorporación de nuevos módulos, el mecanismo es programáticamente.

Modelado: VSEit contiene plantillas ya preparadas que pueden ser utilizadas para modelar, pero aún así, la mayor parte de programación debe ser realizada por los propios usuarios. Está previsto que los usuarios podrán crear *guiones*, a raíz de un enfoque de modelado. VSEit tiene funcionalidades básicas de una GUI, sin embargo, la gama de opciones gráficas debe extenderse en el futuro con el fin de permitir la visualización de modelos más complejos.
Para la generación de modelos en un medio ambiente, es necesario poseer conocimientos de Java. VSEit tiene planes de ofrecer la construcción de bloques que permiten a los usuarios sin conocimientos de programación especificar un mayor número de modelos.

B.22. Zeus

Descripción: [56] Es un framework para la construcción de aplicaciones Multi-Agente colaborativas. Provee un entorno integrado para el desarrollo rápido de sistemas. Define una metodología de diseño de sistemas Multi-Agente y lo soporta mediante un entorno visual para capturar las especificaciones de los mismos. Estas especificaciones son luego utilizadas para generar el código fuente en Java.
El objetivo del proyecto ZEUS es facilitar el desarrollo rápido de nuevas aplicaciones mediante la abstracción de los principios y componentes más comunes a una herramienta. La idea es proveer un framework de propósito general, personalizable, que permita la creación de agentes colaborativos y que pueda ser usada por ingenieros de software con poca experiencia en tecnología de agentes para crear SMA.

Actualización y Mantenimiento: La última versión disponible es del 10 de enero de 2006. Su último release es el 2.0.

Documentación: Es escasa, sólo se encuentra la que se descarga con el framework, que es un javadoc, además de algunos ejemplos.

Lenguaje de programación: Java.

Licencia: MPL [34].

Entorno de simulación: Si bien en la documentación indica que tiene simulación, no se encontró información detallada de la misma.

Mecanismos de extensión: No se encontró información al respecto.

Modelado: El desarrollo se basa en las tareas que realizan los agentes. Como primer paso se definen las ontologías que se utilizarán en el sistema. Una vez que las ontologías están definidas, se especifican las tareas que realizan los distintos agentes

pero sin detallar que tarea corresponde a cada uno. Posteriormente se detallan los agentes y se les asocia a cada uno las tareas que estos deben realizar dentro del sistema. Como último paso se realiza la generación de código, la cual incluye la generación de los agentes y las tareas que estos realizan.

La Tabla B.1 ilustra el resultado de la evaluación de veintisiete frameworks basada en una comparación entre ellos tomando los criterios iniciales de evaluación detallados en el capítulo 3.1.

Framework	**Documentación**	**Lenguaje desarrollo/ especificación**	**Licencia**	**Entorno de Simulación**	**Mecanismo de Extensión**
AgentSheets	Completa	Java/No disponible	Comercial (solo disponible versión trial)	Si	No
AgentTool	Básica	Java/ No disponible	GNU/GPL	No	Si, codificando.
Ascape	Completa	Java/ No disponible	BSD	Si	Si, codificando.
Breve	Básica	Steve, Python / Steve, Python	GPL	Si	Si, a través de plugins
Cormas	Escasa	Smalltalk/ Smalltalk	LGPL	Si	Si, codificando.
ECHO	Ninguna	(Java, C)/ No disponible	Open Source	No disponible	No disponible
Ingenias	Completa	Java/Java	GPL	No	Si, a través de módulos (IDE MARCO – brinda una API)
Jack	Completa	Java/Jack	Freeware (limitada a 60 días)	Si	Si, codificando.
JADE	Completa	Java/Java	LGPL	Si (enfocado a interacciones)	Si, codificando.
Madkit	Básica	Java, Jess, Python, BeanShell/Java, Python	GPL/LGPL	Si, pero se debe acoplar la herramienta TurtleKit	No disponible
MAGSY	Escasa (limitada, a ejemplos en los paquetes de instalación)	Extensión de osp5 Magsy (idioma de Producción)	Libre – Código Cerrado	Si	No disponible
MAML	Básica	C/Lenguaje MAML, puede contener Objetive- C y secuencias Swarm	GNU (El compilador se puede descargar libremente para fines de evaluación)	No	Si, codificando.
MASON	Completa	Java/Java	AFL v3.0	Si	Si, codificando.
Mimosa	Básica	Java/Java script, Jess, Python, Prolog, Smalltalk	LGPL	Si	Si, codificando.
MIMOSE	Básica	Java/Lenguaje propietario	Libre – Código Cerrado	Si	No

NetLogo	Completa	Java/Logo	No Open Source, provee libremente una API para su uso.	Si	Si, codificando.
PS-i	Básica	Tcl/Tk	GPL	Si	No disponible
REPAST	Completa	Java/Java	BSD	Si	Si, codificando.
SeSAm	Completa	Java/ No disponible	LGPL	Si	Si, codificando.
SimAgent	Básica	Pop-11/ No disponible	Open Source	Si	Si, a través de módulos.
SimPack	Completa	C++ / Java	GPL	Si	No disponible
StarLogo	Completa	TNG StarLogo (lenguaje extensión de logo-lenguaje de programación gráfica). Existe una versión Java	Libre – Código Cerrado	Si	No disponible
SugarScape	Escasa	Java/ No disponible	GPL	Si	Si, codificando.
Swarm	Completa	Objective-C, Java/ Objective-C, Java	GPL	Si	Si, a través de módulos.
TeamBots	Básica	Java/ No disponible	Open Source para uso no comercial. (académico e investigación)	Si	No disponible
VSEit	Básica	Java/Java	Se distribuye a pedido. Ponerse en contacto con el desarrollador	Si	Si, codificando.
Zeus	Escasa	Java/ No disponible	Open Source, MPL (Licencia publica de Mozilla)	Si	No disponible

Tabla B.1.: Resumen de frameworks investigados.

Apéndice C
Proceso de desarrollo del editor gráfico

El objetivo de este apéndice, es poder profundizar en el manejo de EMF y GMF para la creación de editores gráficos de modelado específicos de dominio.

C.1. Modelado en EMF

El modelo se define utilizando EMF mediante un lenguaje de definición de modelos denominados .ecore. Todos y cada uno de los subprocesos de GMF están relacionados con dicho modelo, el .ecore constituirá el centro del proyecto en todo momento.

Creación de un modelo de dominio

Se debe crear un proyecto *"Empty EMF Project"*. Se agrega al proyecto EMF un nuevo modelo .ecore_diagram (ver Figura C.1), para esto es necesario estar posicionado en la carpeta *model* del proyecto y se seleccionar *new/other*. En nuestro caso lo llamamos ModeloMA.ecore_diagram y a continuación se da el nombre del archivo de extensión *.ecore*, que suele ser el mismo para mantener la consistencia.

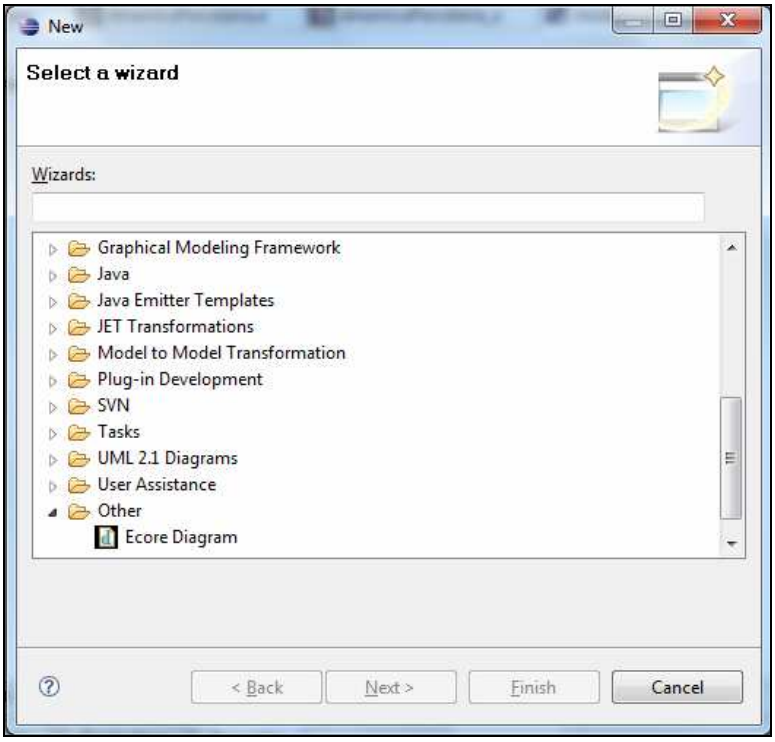

Figura C.1: Creación del diagrama para el modelado

Al crear un diagrama *.ecore*, EMF genera al mismo tiempo un documento XML con extensión *.ecore*, que asocia al diagrama del modelo y que es utilizado para la generación de código.

Se crean los dos archivos y se abre automáticamente el archivo con extensión *.ecore_diagram*. Este archivo presenta una ventana de modelado y una paleta de herramientas como muestra la Figura C.2.

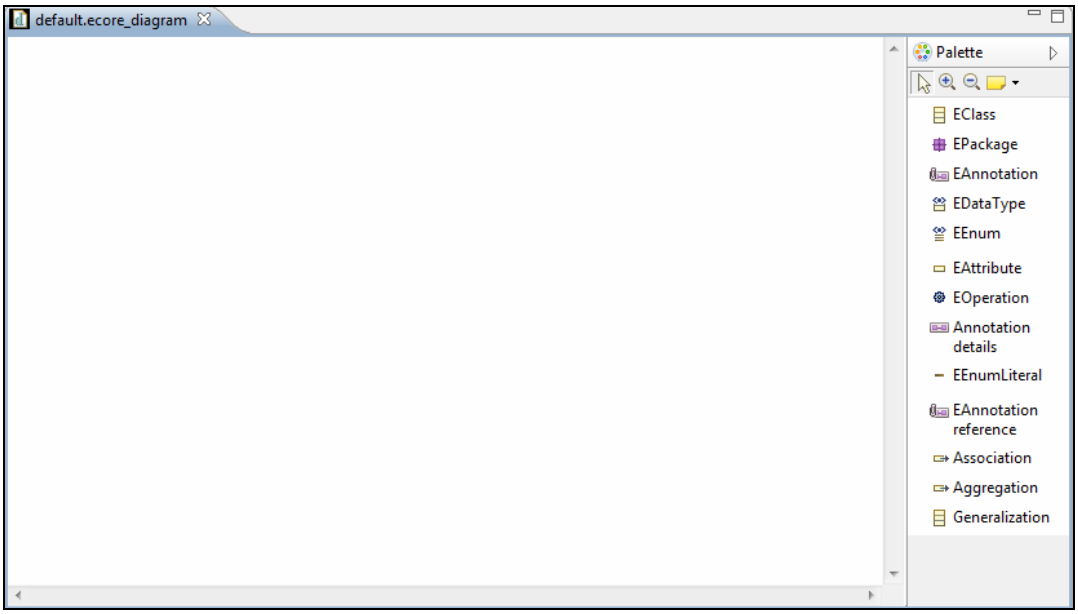

Figura C.2: Ventana de modelado

La paleta de herramientas incluye los diferentes tipos de elementos y relaciones que permiten definir un modelo .ecore. En la paleta se puede distinguir la herramienta *Eclass*, con la que podremos crear nuevas clases, la herramienta *Epackage*, con la que crearemos nuevos paquetes, *EAnnotation*, con la que se crearán anotaciones y comentarios, *EDataType* que permite crear nuevos tipos de datos, *EEnum* que permite crear enumerados, *EAttribute* la cual permite agregar atributos a las clases, *EOperation* que añade operaciones a las clases y por último, las relaciones *Association* que es de asociación, *Aggregation* que es de agregación y *Generalization* que es de herencia.

El modelado se realiza mediante "drag and drop" para cualquiera de las herramientas de la paleta, esto quiere decir que si pulsamos por ejemplo sobre la herramienta de creación de clases y acto seguido pulsamos sobre la ventana de modelado se crea una nueva clase. Por otro lado, es posible añadirles tantos atributos y operaciones como se deseen. Hay dos posibilidades de hacerlo: una de ellas consiste en arrastrar desde la paleta de modelado y soltar sobre la clase los atributos y operaciones y la otra es a través de un menú emergente que aparece superponiendo el puntero sobre la clase en cuestión.
Las relaciones entre clases se instancian gráficamente mediante las 3 últimas herramientas que aparecen en la paleta de modelado. Se ha de tener en cuenta, que las asociaciones en EMF son siempre unidireccionales. Si se desea modelar una relación bidireccional, se realiza mediante dos asociaciones que relacionen las dos mismas clases en sentidos opuestos y fusionarlas en una sola bidireccional. Para ello, se ha de seleccionar dentro de una asociación, su opuesta en la propiedad *Eopposite*.

Para definir el tipo de un atributo, clase o relación es necesario editar sus propiedades. Esto se realiza pulsando con el botón derecho encima del elemento del que se desean ver las propiedades y seleccionar la opción *Show Properties View*. A continuación, aparece una ventana en la parte inferior del entorno de Eclipse mediante la que se pueden editar todas las propiedades de cada elemento. Entre las propiedades que definen a una clase están el nombre, si pertenecen a alguna subclase, si son de algún tipo predefinido, entre otras. Para editar el tipo de los atributos, se ha de pinchar en la propiedad *EType* y seleccionar el tipo de datos adecuado para cada atributo.
También, podemos crear enumeraciones de elementos, es decir, listas de tipos de datos. Esta definición se realiza sobre la ventana de modelado en el que definimos nuestro modelo .ecore. Para ello añadimos un objeto de tipo *EEnum* de la barra de herramientas a la ventana de modelado mediante "drag and drop". Posteriormente, le damos un nombre en el campo *Name*. Este será el nombre que tendrá nuestro tipo de datos enumerado. Para crear la lista sólo nos queda definir dentro del elemento *EEnum* tantos *EEnumLiteral* como número de tipos de datos queramos introducir. *EEnumLiteral* se encuentra también en la barra de herramientas de modelado. A cada *EEnumLiteral* se le ha de asociar su nombre, que será el nombre con el que aparecerá en la enumeración. Para ello, introducimos el mismo nombre para el campo *Name* y Literal del *EEnumLiteral*.

Dado que EMF expresa en formato XML el modelo correspondiente a un diagrama .ecore, conviene resaltar que para definir un modelo correctamente en EMF es necesario incluir un elemento raíz (una clase) que relacione mediante agregación todas las clases del modelo, en nuestro caso de estudio la clase Metamodelo agrega todas las restantes clases.

Cuando se crea una relación se le asigna un nombre y se especifica su cardinalidad. Para esto último, existen dos propiedades dentro de la vista de propiedades, *Lower bound* en la cual se especifica la cardinalidad mínima, y *Upper bound*, en la que se especifica la cardinalidad máxima. Cabe destacar que para conseguir la cardinalidad N, se debe poner el valor -1 y automáticamente, al dejar de editar la relación, se cambia el valor a *.

En éste proyecto fue necesario tener dos conjuntos de editores gráficos diferentes, uno para el diagrama de clase y otro para el diagrama de actividad, por lo cual se generó dos .ecore con un único .ecore_diagram, dado que presenta el mismo modelo.

En la Tabla C.1 que se muestra a continuación, se puede apreciar las propiedades a modificar en cada uno de los modelados, dichas propiedades corresponden al paquete en el cual se encuentran, donde "*name*" corresponde al nombre del modelo, "*prefix*" al nombre que se utiliza de forma predeterminado cuando se serializan instancias del paquete en las clases, "*URI*" corresponde a la identificación URI del paquete.

Modelo de Diagrama de Clase	
Name	AgroEcoMA
Ns prefix	AgroEcoMA
Ns URI	http://AgroEcoMA
Modelo Diagrama de Actividad	
Name	AgroEcoMAD
Ns prefix	AgroEcoMAD
Ns URI	http://AgroEcoMAD

Tabla C.1: Seteo de propiedades

C.2. Creación del editor gráfico con GMF

GMF es un framework para la generación de editores gráficos basados en EMF y GEF. Este último es el que permite al desarrollador crear un editor gráfico completo de forma rápida a partir de un modelo.

Creación de una paleta gráfica

Se crea un nuevo proyecto seleccionando "*New GMF Project*" de la carpeta *Graphical Modeling Framework.*

Como necesitamos tener dos paletas gráficas fue necesario crear dos proyectos GMF, uno que permite modelar diagramas de clase y otro que permite crear diagramas de actividad.

A lo largo de este apéndice se tomara solo el correspondiente al diagrama de clase y si se ve la necesidad de detallar algún punto en particular para el diagrama de actividad se detallará.

Importación del modelo Ecore al proyecto GMF

En el proyecto GMF se selecciona "*New EMF Model*" que se encuentra en la carpeta *Other*, se crea un archivo de extensión *.genmodel*, se elige la opción *Ecore Model* con el modelo de datos, creado previamente.
En caso de crear un modelo desde cero desde GMF, el procedimiento es el mismo, pero creándose un *Ecore Model*.

Generación de código del modelo

El archivo de extensión *.genmodel*, permite transformar automáticamente el modelo .ecore que hemos definido a código fuente. El código se genera aplicando patrones de transformación. El resultado de la generación es un conjunto de clases Java, que serán utilizadas más adelante en el proceso de creación del Framework de modelado, para que todos los elementos que lo constituyen se comporten tal y como el modelo .ecore establece.
Antes de generar el código del modelo es preciso configurar el modelo *.genmodel* para especificar su paquete base. Para ello hacemos doble clic en el archivo de

extensión *.genmodel*, se abre una ventana con un árbol similar al .ecore. Presionamos con el botón derecho sobre el paquete morado, de nombre *AgroEcoMA* y seleccionamos *Show properties view*. Se nos abre una tabla que debemos rellenar como indica la Figura C.3.

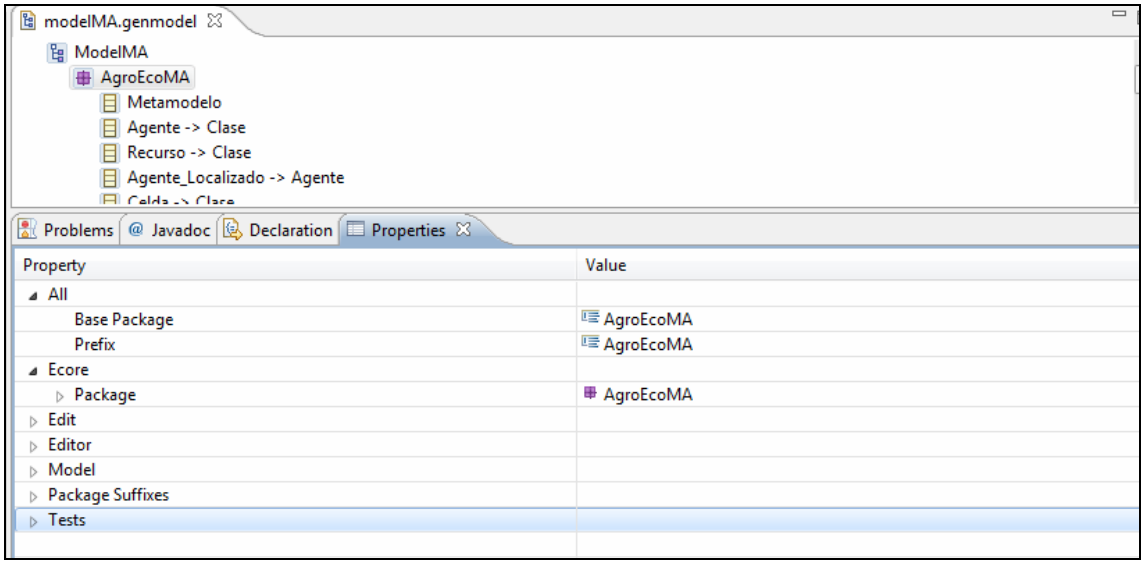

Figura C.3: Seteo de propiedades para el archivo .genmodel

El siguiente paso es generar el código del modelo. Pulsamos con el botón derecho sobre el paquete morado de nombre *AgroEcoMA* y seleccionamos *Generate Model Code*. Se generará automáticamente un plugin con el código del modelo. A continuación, realizamos el mismo proceso seleccionando *Generate Edit Code* y *Generate Editor Code*. Este último será el intérprete de Eclipse que nos permita ver nuestro modelo específico de dominio en forma de árbol tal y como nos muestra los documentos .ecore y cualquier componente de GFM.

Definición gráfica del modelo

La definición gráfica del modelo consiste en definir el aspecto que van a tener las primitivas de modelado en nuestro editor gráfico. Así, GMF de una forma automática, nos permite personalizar el aspecto de la aplicación de construcción de modelos específicos de dominio a nuestro gusto.

Para crear la definición grafica del modelo seleccionamos un nuevo archivo del tipo "*Simple Graphical Definition Model*" que se encuentra en la carpeta *Graphical Modeling Framework*.
El archivo generado es de extensión *.gmfgraph*, es necesario seleccionar el archivo .ecore que tomará como base y que se quiere representar de forma gráfica, en la ventana que le continua se debe seleccionar la entidad que es contenedora de toda la realidad a modelar, en este caso es Metamodelo.
En este paso aparece una ventana similar a la Figura C.4, donde se deberá elegir que elementos del modelo de dominio actuarán como nodos, cuales como enlaces y cuales como etiquetas.

En la ventana se visualiza una tabla de tres columnas en la que se encuentran colocados de izquierda a derecha y están representados gráficamente en la cabecera de la siguiente forma: cuadrado con rayas horizontales (*nodos*), línea inclinada (*enlaces*), y letra A (*etiquetas*).

Cada elemento del modelo tiene asociada una casilla de selección para cada columna de la tabla que es posible seleccionar, para así elegir el tipo de representación gráfica que se desea tener. Parte de la selección se puede ver en la Figura C.4.

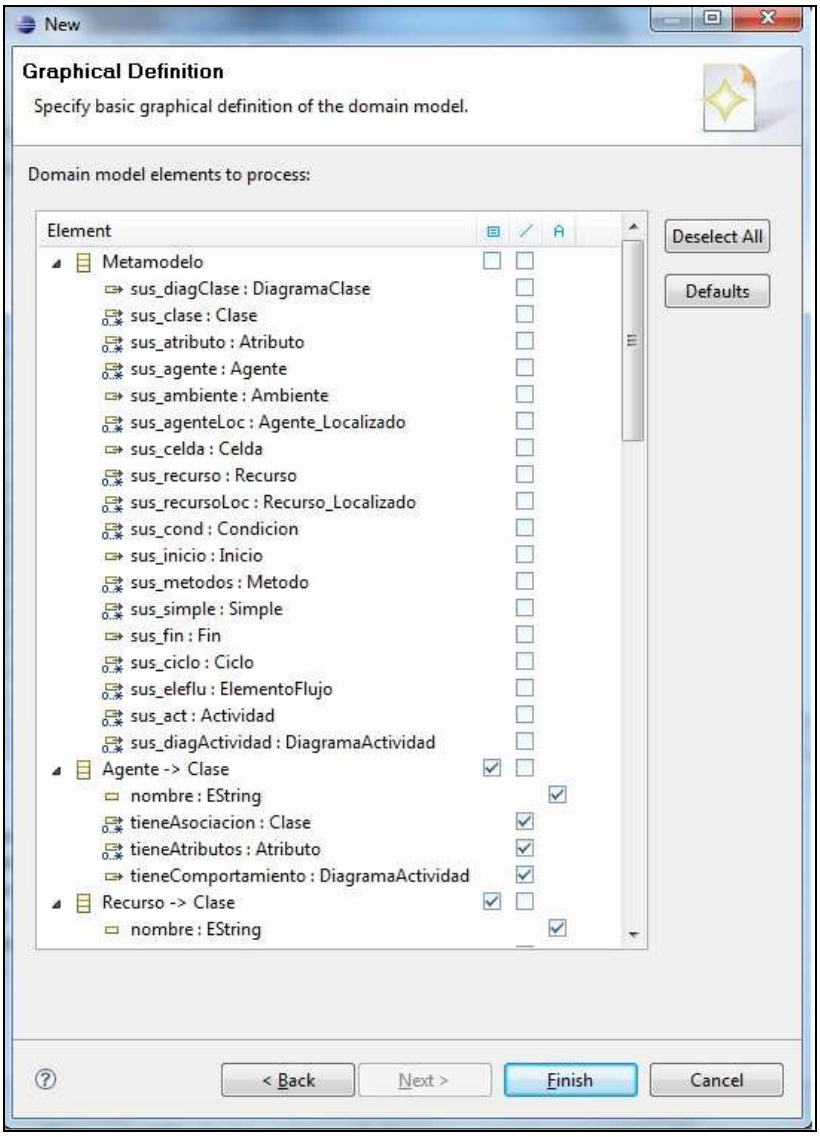

Figura C.4: Definición gráfica del modelo

Para poder ejemplificar se muestra en la Tabla C.2 la definición que se usó, para el proyecto correspondiente al diagrama de clase.

	Nodo	Enlace	Atributo
ClaseTieneAsociacion		X	
ClaseTieneAtributo		X	
Agente	X		
AgenteNombre			X
Recurso	X		
RecursoNombre			X
AgenteLocalizado	X		
AgenteLocalizadoNombre			X
RecursoLocalizado	X		
RecursoLocalizadoNombre			X
Celda	X		
CeldaNombre			X
Ambiente	X		
AmbienteNombre			X
Atributo	X		
AtributoNombre			X
AtributoTipo			X
DiagramaClase	X		
DiagramaClaseNombre			X

Tabla C.2: Definición gráfica por entidad

Una vez definido todos los elementos obtenemos como resultado la estructura que se muestra en la Figura C.5.

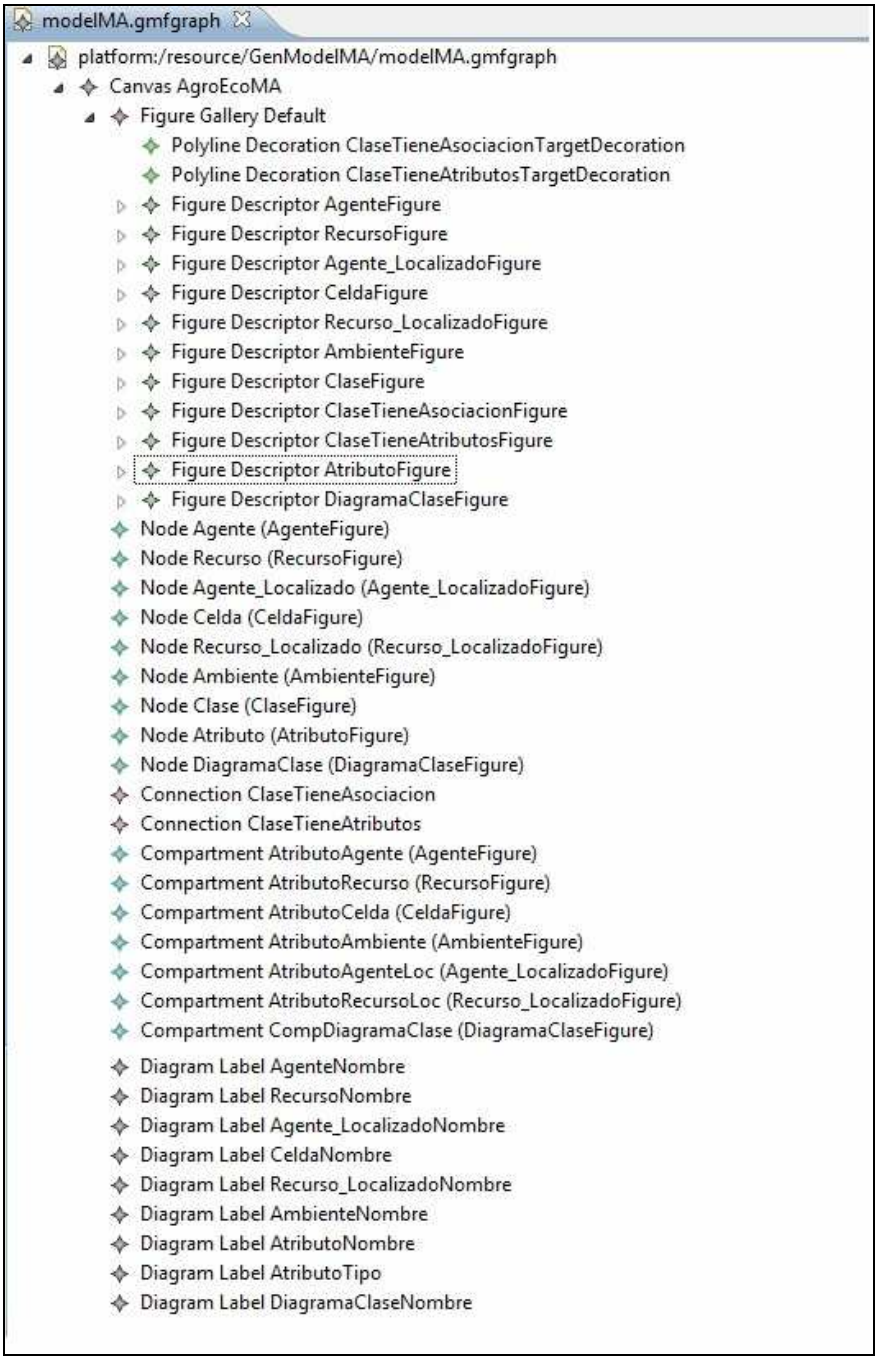

Figura C.5: Estructura del archivo .gmfgraph

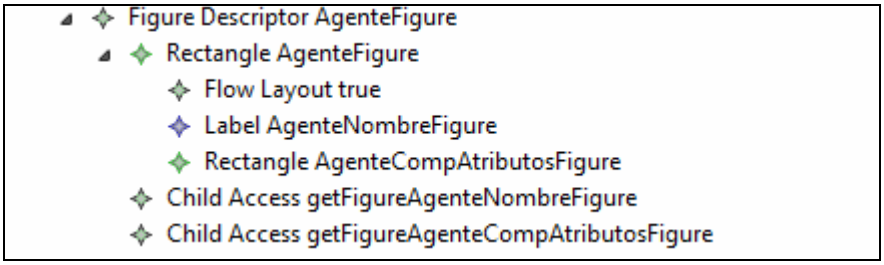

Figura C.6: Definición gráfica de la entidad *Agente*

Al desplegar por ejemplo, Agente como se muestra en la Figura C.6, podemos ver como se visualizará en la herramienta, si nos centramos en la entidad agente, podemos ver que el dibujo de la entidad es un rectángulo (*Rectangle AgentFigure*) que contiene un atributo nombre (*Label AgentNombreFigure*) y otro rectángulo (*RectangleAgentCompAtributosFigure*) en los cuales se van a encontrar los atributos que se quieran definir.

El resto de las entidades se comportan de manera similar a excepción de la entidad DiagramaClase, la cual se puede ver en la Figura C.7, que tiene la forma de un rectángulo (*Rectangle DiagramaClaseFigure*), tiene una etiqueta llamada nombre (*Label DiagramaClaseNombreFigure*) y un compartimiento donde se insertaran todos los elementos pertenecientes al diagrama de clase (*Rectangle DiagramaClaseCompFigure*), como son entidades, enlaces y etiquetas.

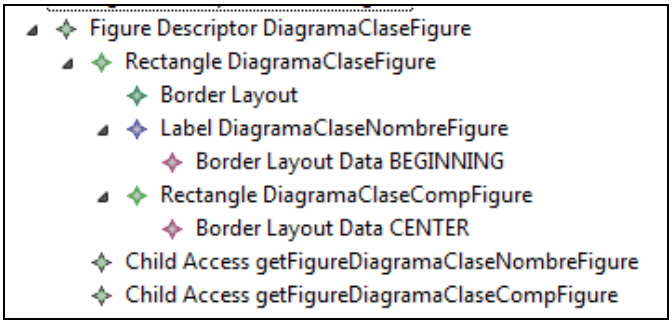

Figura C.7: Definición gráfica de la entidad *DiagramaClase*

Definición de la paleta de modelado

Se define la paleta de herramientas con la que construiremos los modelos. Para ello, se ha de seleccionar "*Simple Tooling Definition Model*" de la carpeta *Graphical Modeling Framework*.

La definición de la paleta de herramientas tiene extensión *.gmftool.*
Para esto es necesario seleccionar el modelo .ecore definido y elegir la entidad contenedora (Metamodelo) de la realidad que se está modelando.
Se presentan los elementos del modelo y nos permite elegir aquellos que van aparecer como nodos o como enlaces en la barra de herramientas, ver Figura C.8.
A continuación se muestra la Tabla C.3 donde se indica la selección realizada.

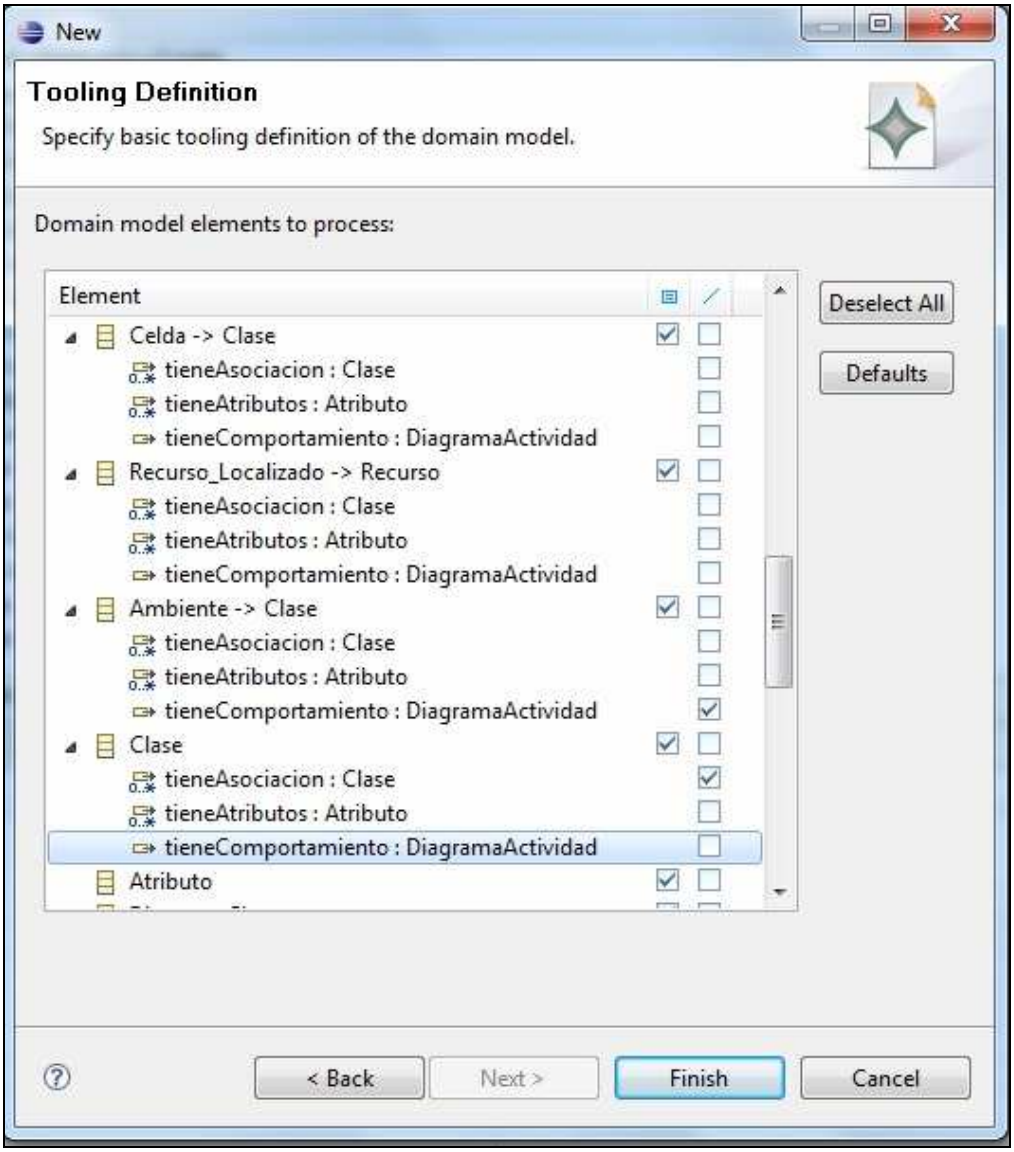

Figura C.8: Definición gráfica de la paleta de herramientas

Elemento	Entidad	Asociación
Agente	X	
Recurso	X	
Agente_Localizado	X	
Recurso_Localizado	X	
Celda	X	
Ambiente	X	
Diagrama Clase	X	
Atributo	X	
Asociacion		X

Tabla C.3: Definición gráfica de la paleta de herramientas por entidad

Finalmente la definición de paleta para el diagrama de clase se puede visualizar en la Figura C.9.
Por cada elemento que se muestra en la paleta se pueden seleccionar dos imágenes una corresponde a la imagen con la cual se ve el elemento en el paleta grafica y la otra corresponde a la imagen con la cual se ve una vez volcado el elemento en el editor propio. Dichas imágenes se pueden modificar indicando la ruta en la cual se encuentran las imágenes, de no hacerlo quedan las imágenes que se insertan por defecto.
Las imágenes se deben de guardar en el archivo de iconos del proyecto generado de tipo edit.

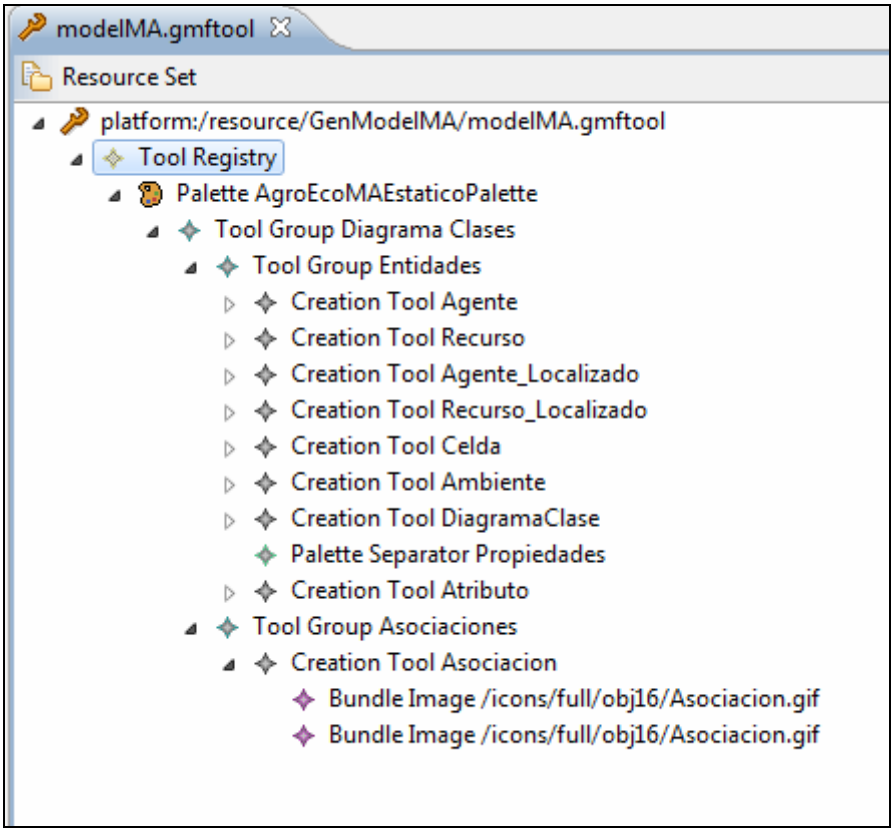

Figura C.9: Definición de la paleta de herramientas para el diagrama de clases

Definición de Mapeos

El proceso de mapping es donde se une todo lo previamente creado para formar la herramienta de construcción de modelos específicos de dominio.
Se crea un archivo de extensión *.gmfmap* que se obtiene seleccionando "*Guide Mapping Model Creation*" de la carpeta *Graphical Modeling Framework*.

Se debe seleccionar el modelo .ecore, el elemento raíz de nuestro modelo (Metamodelo), y los archivos *.gmftool* y *.gmfgraph* construidos previamente.

A continuación aparece una ventana en la cual se especifican los enlaces y links que se mapearan como se muestra en la Figura C.10.

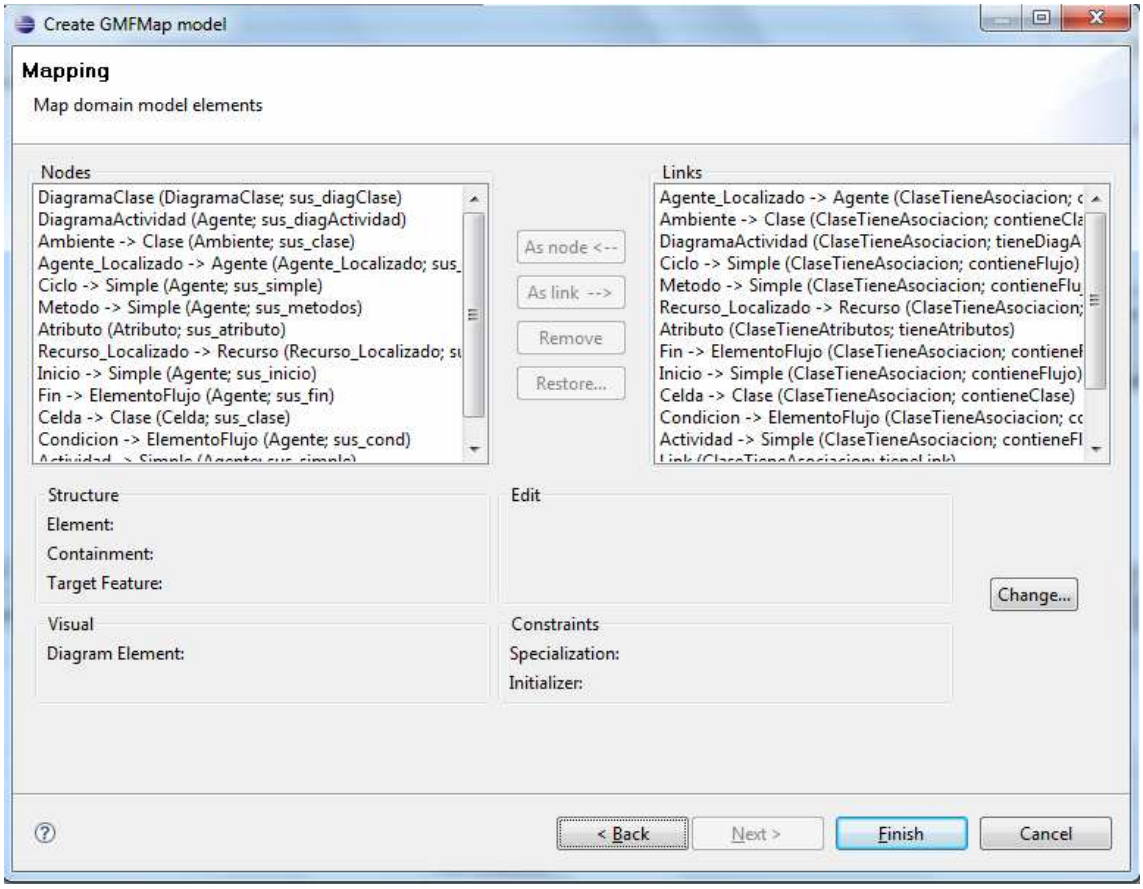

Figura C.10: Mapeo del modelo

Por cada entidad que aparece en *Nodes* se selecciona *Change* y se modifica con los criterios definidos previamente para esas entidades.
Una vez completado este paso se puede ver todo en la estructura de árbol que se muestra en la Figura C.11.

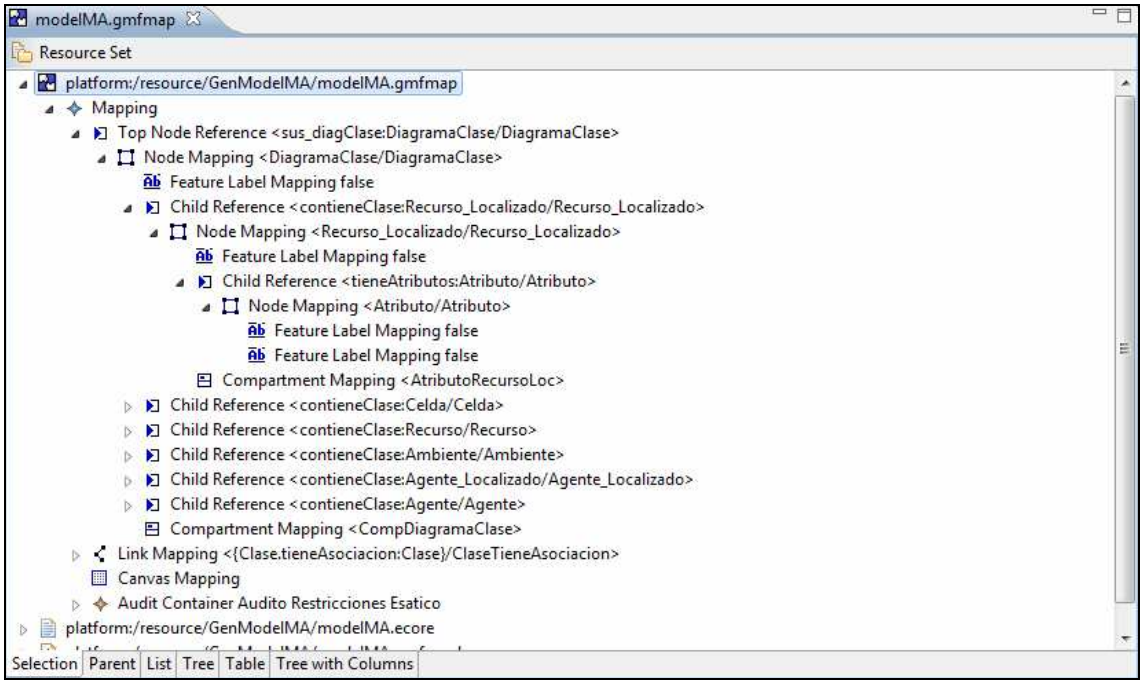

Figura C.11: Estructura del archivo *.gmfmap*

Es probable que las etiquetas no se creen automáticamente por lo cual es necesario declararlas nuevamente en éste paso, dentro de las entidades a las cuales pertenecen, seleccionando en *node mapping* un elemento "*Feature Label Mapping*".

En este archivo es donde se definen las reglas de tipo ocl que validarán las restricciones declaradas para el modelo.
Se crean dentro de un "*Audit Container*" y se visualizan en la Figura C.12.

En las propiedades se puede ver el cuerpo de la sentencia y el lenguaje en el cual se declaran.
También se definió una regla para el link por medio de "*Links Constraints*".

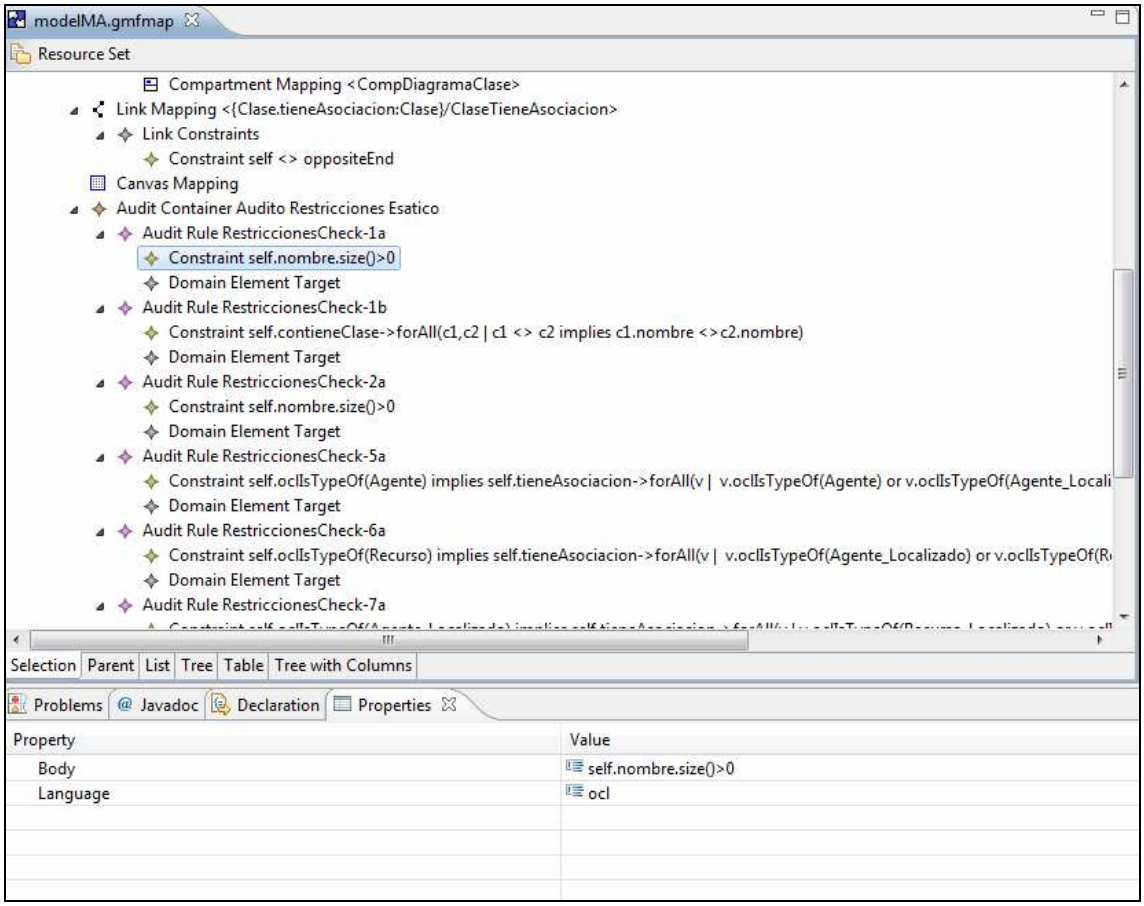

Figura C.12: Reglas para la validación de modelos

Generación de código

Una vez completado todos los pasos previamente explicados es necesario generar su código.

Si presionamos con el botón derecho en el archivo *.gmfmap* generado previamente y seleccionamos "*Create Generator Mode*", introducimos el nombre del generador, quedando un archivo de extensión *.gmfgen*. Es necesario indicar los archivos *.gmfmap* y *.genmodel* a los cuales referencia.

Fue necesario modificar las propiedades del archivo generado (*.gmfgen*) en la sección *EditPart*, indicando en la propiedad *Diagram* las condiciones de *Validation Decorators* y *Validation Enabled* en "*true*", además de indicar en *Providers* que *Provider Priority* tendrá el valor "*Medium*", para permitir que las OCL declaradas en el mapeo funcionen correctamente.

Finalmente, pulsamos con el botón derecho sobre *.gmfgen* y seleccionamos "*Generate Diagram Code*". Si se ha generado correctamente aparecerá un mensaje de confirmación y se genera una carpeta correspondiente a *diagram*.

Se realizan los mismos pasos para la generación del editor gráfico correspondiente al diagrama de actividad de nombre *AgroEcoMAD*.

C.3. Restricciones al modelo

A continuación se especifican las restricciones para el modelo definido en la sección 4.2.1.

Clase:

- La clase debe tener un nombre, el cual no puede ser nulo.
- El nombre de la clase lo identifica, no puede haber repetidos.
- Una clase no debe tener una asociación con sigo misma.

DiagramaClase:

- El diagrama de clase debe tener un nombre, el cual no puede ser nulo.
- En el diagrama de clase se puede a lo sumo definir una entidad *Celda*.
- En el diagrama de clase debe haber una sola entidad *Ambiente*.

Agente:

- Un agente podrá tener asociaciones con otras entidades *Agente*, *Recurso*, *Agente_Localizado*, *Recurso_Localizado*.

Recurso:

- Un recurso podrá tener asociaciones con otras entidades *Recurso*, *Agente_Localizado*, *Recurso_Localizado*.

Agente_Localizado:

- Un agente localizado podrá tener asociaciones con otras entidades *Agente_Localizado*, *Recurso_Localizado* y *Celda*.

Recurso_Localizado:

- Un recurso localizado podrá tener asociaciones con otras entidades *Recurso_Localizado* y *Celda*.

ElementoFlujo:

- Las sub-entidades de *ElementoFlujo* (*Actividad*, *Método*, *Ciclo*, *Condición*) deben tener un nombre, el cual no puede ser nulo.
- Las sub-entidades de *ElementoFlujo* no deben tener una asociación con sigo mismas.
- Las sub-entidades de *ElementoFlujo* tienen asociaciones con entidades que pertenecen al mismo diagrama de actividad.

Diagrama_Actividad:

- Un diagrama de actividad debe tener un nombre, el cual no puede ser nulo.
- El nombre del diagrama de actividad lo identifica, no puede haber repetidos.

Condición:

- Una condición tiene como máximo dos asociaciones, una cuando evalúa en *true* y otra cuando evalúa en *false*, pero es necesario que siempre exista por lo menos una (correspondiente a t*rue*).

Simple:

- De una sub-entidad *Simple* siempre debe existir una sola asociación saliente.

Fin:

- De una entidad *Fin* no deben existir asociaciones salientes.

Apéndice D

Plantillas de Acceleo

En esta sección se detallan los templates más relevantes utilizados en el módulo de generación de código.

D.1. Estructura de los templates

En un template Acceleo se pueden identificar tres secciones principales:

- En la primera sección se realiza la importación del modelo de dominio, el cual proporciona el acceso a las entidades definidas, posibilitando la navegación entre sus atributos y relaciones.
- En la segunda sección se define el script principal del template, donde se identifica que elemento del modelo de dominio será utilizado como base para navegar por el modelo, así como el nombre del archivo de salida resultado de la ejecución del script.
- En la tercera sección se especifican las reglas de transformación específicas al elemento del modelo de dominio mencionado en la primera sección.

D.2. Template Agent.mt

Este template genera la clase que representa a la entidad Agente. En dicha clase se especifican los atributos asignados así como el código del comportamiento definido.

A continuación se presenta el template dividido en las secciones anteriormente mencionadas, con la salvedad de que la sección tres se subdividió para una mejor comprensión de su estructura.

Sección 1:

```
<%
metamodel /agroeco.mas.code.gen/model/modelMA.ecore
%>
```

Se realiza la importación del modelo de dominio definido mediante el framework EMF explicado en el capítulo 4.

Sección 2:

```
<%script type="AgroEcoMA.Agente" name="Agente"
file="<%nombre%>.java"%>
```

En la primera línea se especifica el elemento de dominio a considerar, en este caso **AgroEcoMA.Agente** y la segunda línea corresponde al tipo de archivo a generar donde

el nombre corresponde al valor especificado en la propiedad *nombre* del elemento correspondiente.

Sección 3.1:

```
package ascape;

import org.ascape.model.CellOccupant;

<%if (nombre.length() != 0){%>
public class <%nombre%> extends CellOccupant {
   <%if (tieneAtributos.nSize == 0){%>
      //No attributes.
      <%}else{%>
      <%for (tieneAtributos) {%>
      <%if (tipo.length() != 0 && nombre.length() != 0){%>
    private <%tipo%> <%nombre%>;
    <%}%>
      <%}%>
      <%}%>

      public void scapeCreated() {
            //Aplico reglas
            getScape().addRule(UPDATE_RULE);
      }

      public void update(){
            this.step();
      }

      /*
       * metodo customizable, se define la logica del agente
       */
      public void step (){
      <%tieneComportamiento.contieneFlujo.nGet(0).default%>
      }

      <%for (tieneAtributos) {%>
    <%if (tipo.length() != 0 && nombre.length() != 0){%>
    public void set<%nombre.toU1Case()%>(<%tipo%> <%nombre%>){
      this.<%nombre%> = <%nombre%>;
    }

    public <%tipo%> get<%nombre.toU1Case()%>(){
      return this.<%nombre%>;
    }
      <%}%>
      <%}%>

}
<%}%>
```

Se define el paquete en el cual se generará el .java, se especifican las importaciones necesarias así como el cuerpo de la clase el cual contiene la definición de los atributos especificados para la entidad. Para acceder a los atributos basta con navegar por la asociación *tieneAtributos* e iterar sobre la colección para transformar la entidad *Atributo* en atributos de clase java especificando tipo y nombre.
También para cada atributo se especifican los métodos *get* y *set* correspondientes.

Sección 3.2:

```
<%script type="AgroEcoMA.Inicio" name="default"%>
<%tieneLink.tieneElemFlujo.default%>
<%script type="AgroEcoMA.Actividad" name="default"%>
//<%nombre%>;
<%tieneLink.tieneElemFlujo.default%>
<%script type="AgroEcoMA.Ciclo" name="default"%>
while (true){
<%tieneDiagAct.contieneFlujo.tieneLink.tieneElemFlujo.default%>
}
<%script type="AgroEcoMA.Condicion" name="default"%>
<%if (tieneLink.nSize == 2){%>
<%if (tieneLink.nGet(0).nombre.startsWith(true)){%>
if (true){ //<%expresion%>
    <%tieneLink.nGet(0).tieneElemFlujo.default%>
}else{
    <%tieneLink.nGet(1).tieneElemFlujo.default%>
}
<%}else{%>
if (true){ //<%expresion%>
    <%tieneLink.nGet(1).tieneElemFlujo.default%>
}else{
    <%tieneLink.nGet(0).tieneElemFlujo.default%>
}
<%}%>
<%}else{%>
<%if (tieneLink.nGet(0).nombre.startsWith(true)){%>
if (true){ //<%expresion%>
    <%tieneLink.nGet(0).tieneElemFlujo.default%>
}
<%}else{%>
if (true){ //!<%expresion%>
    <%tieneLink.nGet(0).tieneElemFlujo.default%>
}
<%}%>
<%}%>
<%script type="AgroEcoMA.Metodo" name="default"%>
//<%nombre%>();
<%tieneLink.tieneElemFlujo.default%>
<%script type="AgroEcoMA.Fin" name="default"%>
```

En el método step se invocan los script definidos en la sección 3.B, para esto se navega por la asociación *tieneComportamiento*, accediendo a la entidad *DiagramaActividad* la cual tiene asociada una colección de *ElementoFlujo*, de esta colección se toma el primer elemento el cual corresponde a la entidad *Inicio* y para este elemento se invocan los script llamados default. Luego a partir de las entidades asociadas a la entidad *Inicio*, se va invocando a los script default de tal forma que al coincidir cada entidad con el script especifico dado el campo type se genera la sintaxis java correspondiente.

D.3. Template Model.mt

Este template genera la clase java Model, la cual es la responsable de definir las estructuras Ascape necesarias para la ejecución de la simulación así como la integración de las diferentes clases definidas.
Dada la diversidad de conceptos definidos en el template la sección tres es dividida en varias sub-secciones.

Sección 1:

```
<%
metamodel /agroeco.mas.code.gen/model/modelMA.ecore
%>
```

Se realiza la importación del modelo de dominio específico.

Sección 2:

```
<%script type="AgroEcoMA.Metamodelo" name="Model" file="Model.java"%>
```

Se define el script, mediante el cual se especifica el elemento de dominio a considerar **AgroEcoMA.Metamodelo** y el nombre de la clase java a generar Model.java.
A partir del elemento **AgroEcoMA.Metamodelo** se puede navegar por todas las entidades definidas en el modelo de dominio.

Sección 3.1:

```
package ascape;

import org.ascape.model.Scape;
import org.ascape.model.event.ScapeEvent;
import org.ascape.model.space.Array2D;
import org.ascape.model.space.Array2DMoore;
import org.ascape.view.vis.Overhead2DView;
import org.ascape.model.HostCell;

public class Model extends Scape {
      private Scape grilla;
      <%for (sus_diagClase.contieneClase) {%>
      <%if (trim().startsWith("AgroEcoMA.Ambiente")){%>
      private <%nombre%> miAmbiente;
      <%}%>
      <%}%>
      //Para la definicion de la vista
      private Overhead2DView overheadView;
      //Lista de objetos que froman parte del modleo, agente, agentes
      localizado,recurso,  recurso localizado
      <%for (sus_diagClase.contieneClase) {%>
      <%if (trim().startsWith("AgroEcoMA.Agente") ||
       trim().startsWith("AgroEcoMA.Agente_Localizado") ||
       trim().startsWith("AgroEcoMA.Recurso") ||
       trim().startsWith("AgroEcoMA.Recurso_Localizado") ){%>
       private Scape <%nombre.toL1Case()%>Ocupants;
      <%}%>
      <%}%>
```

Se especifican las importaciones de las estructuras Ascape necesarias, se define el cuerpo de la clase java a generar así como algunos atributos globales. Dentro de estos atributos globales se especifican las entidades (agentes o recursos) que formaran parte del modelo.

Sección 3.2:

```
public void createScape() {
   super.createScape();
   //Se crea una instancia de ambiente
   <%for (sus_diagClase.contieneClase) {%>
   <%if (trim().startsWith("AgroEcoMA.Ambiente")){%>
   miAmbiente = new <%nombre%>();
   <%}%>
   <%}%>
   //Moore la celda tiene 8 vecinos
   //VonNeumann la celda tiene 4 vecinos
   grilla = new Scape(new Array2DMoore());
   grilla.setName("<%sus_diagClase.nombre%>");
   grilla.setPrototypeAgent(new HostCell());
   boolean grillaSize = false;
   <%for (sus_diagClase.contieneClase) {%>
   <%if (trim().startsWith("AgroEcoMA.Ambiente")){%>
   grillaSize = true;
  ((Array2D)grilla.getSpace()).setExtent(miAmbiente.getAnchoGrilla(),
   miAmbiente.getLargoGrilla());
   <%}%>
   <%}%>
   if (!grillaSize){
        ((Array2D) grilla.getSpace()).setExtent(10,10);
   }
   add(grilla);
   //Creo los ocupantes de la grilla
   <%for (sus_diagClase.contieneClase) {%>
   <%if (trim().startsWith("AgroEcoMA.Agente_Localizado") ||
         trim().startsWith("AgroEcoMA.Recurso_Localizado")){%>
    <%nombre.toL1Case()%>Ocupants = new Scape();
    creoOcupants<%nombre.toU1Case()%>(<%nombre.toL1Case()%>Ocupants);
  <%}%>
 <%}%>
 //Creo los objetos no localizables
 <%for (sus_diagClase.contieneClase) {%>
 <%if (trim().startsWith("AgroEcoMA.Agente") ||
       trim().startsWith("AgroEcoMA.Recurso")){%>
 <%nombre.toL1Case()%>Ocupants = new Scape();
 creoOcupants<%nombre.toU1Case()%>(<%nombre.toL1Case()%>Ocupants);
<%}%>
<%}%>
}
```

Se especifica el método *createScape* el cual es invocado automáticamente por Ascape y tiene como responsabilidad crear el elemento *Scape* que representa la grilla, la cual es una matriz de dos dimensiones, así como la creación de las entidades que interaccionaran en el modelo especificado.

Sección 3.3:

```
<%for (sus_diagClase.contieneClase) {%>
      <%if (trim().startsWith("AgroEcoMA.Agente_Localizado") ||
trim().startsWith("AgroEcoMA.Recurso_Localizado")){%>
public void creoOcupants<%nombre.toU1Case()%>(Scape ocupantes){

   <%nombre%> unOcupante = new <%nombre%>();
   unOcupante.setHostScape(grilla);
   ocupantes.setPrototypeAgent(unOcupante);
   add(ocupantes);

}
<%}%>
<%}%>
```

Se define el método *creoOcupants*, el cual tiene la responsabilidad de crear los prototipos de entidades que van a ocupar un lugar en la grilla definida en la sección 3.4. Esto se indica asignando al prototipo definido su contenedor, que en este caso es la instancia de la clase *Scape* grilla.

Sección 3.4:

```
<%for (sus_diagClase.contieneClase) {%>
      <%if ((trim().startsWith("AgroEcoMA.Agente") &&
!trim().startsWith("AgroEcoMA.Agente_Localizado")) ||
(!trim().startsWith("AgroEcoMA.Recurso_Localizado") &&
trim().startsWith("AgroEcoMA.Recurso")){%>
      public void creoOcupants<%nombre.toU1Case()%>(Scape ocupantes){

        <%nombre%> unOcupante = new <%nombre%>();
        ocupantes.setPrototypeAgent(unOcupante);
        add(ocupantes);
    }
      <%}%>
      <%}%>
```

Se define el método *creoOcupants* para las entidades que no son localizables, a diferencia de lo explicado en la sección 5 de este apéndice, no se asigna un contenedor a estos prototipos.

Sección 3.5:

```
public void createGraphicViews() {
      super.createGraphicViews();
      overheadView = new Overhead2DView();
      grilla.addView(overheadView);
}
```

Se especifica el método *createGraphicViews* el cual es invocado automáticamente por Ascape y tiene como responsabilidad asignarle una representación gráfica a la grilla definida.

Apéndice E
Manual de Usuario

El propósito de este anexo es brindarle al usuario final, una explicación detallada del uso de FW4SimSMA, de forma tal que le sirva como guía para realizar la definición de sus propios modelos. Como base para la creación del manual, se utilizó la información resultante de la implementación del caso de estudio presentado en el capítulo 5.

En el capítulo 4 se describe el proceso completo para realizar el modelado y la ejecución de una simulación de una realidad específica mediante el uso de FW4SimSMA. Este proceso es utilizado como marco para la explicación del uso de FW4SimSMA.

E.1. Creación del proyecto

El primer paso consiste en la creación de un proyecto java en el IDE Eclipse.
La Figura E.1 ilustra el resultado de la creación del proyecto llamado CASO-ESTUDIO.

Figura E.1: Proyecto creado

E.2. Definición de la estructura estática

Se continúa con la creación del diagrama de estructura estática en el cual se especifican las entidades y relaciones presentes en la realidad a modelar. Para crear el diagrama se debe hacer click derecho sobre el proyecto generado, posteriormente en el menú ir a *New/Example.* Luego se debe seleccionar *ModelMA Diagram* tal como se muestra en la Figura E.2 e ingresar el nombre del diagrama.

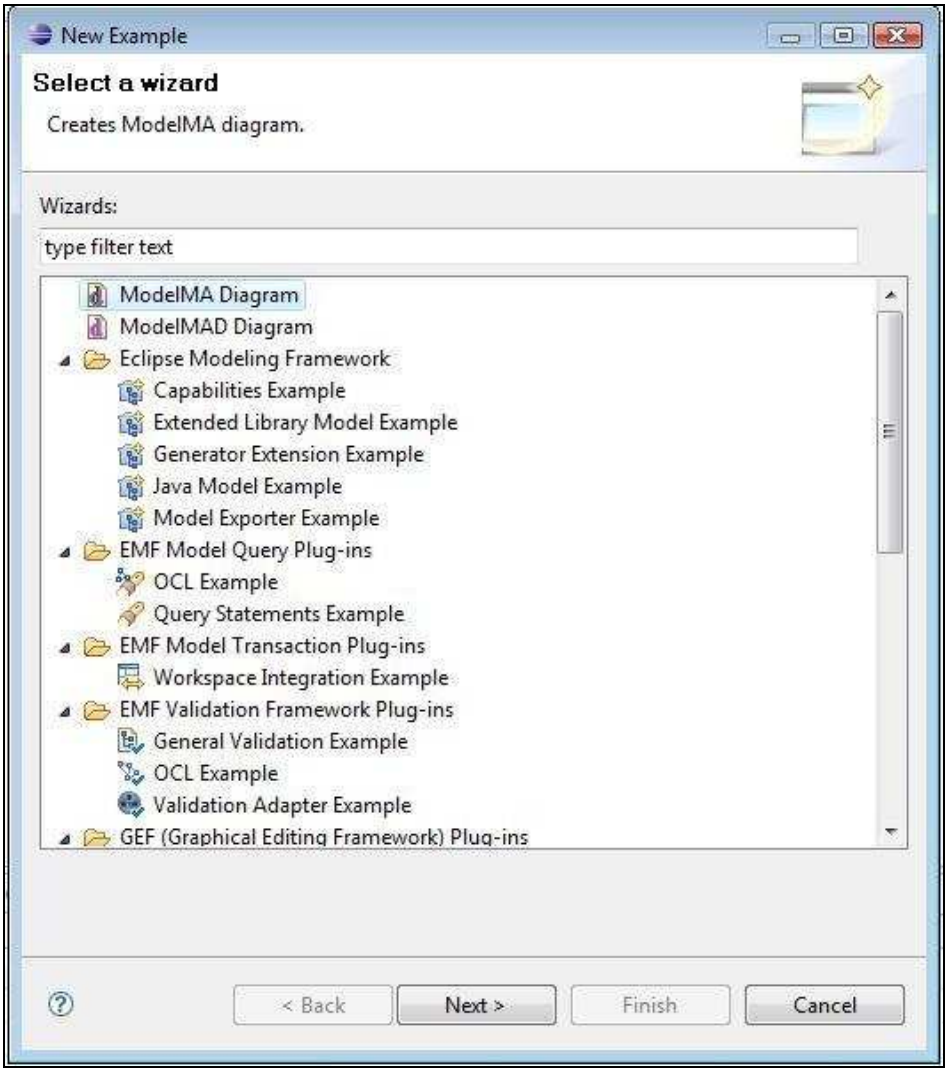

Figura E.2: Selección diagrama de estructura estática

Una vez creado el diagrama se abre automáticamente el editor gráfico, proporcionando a la izquierda de la pantalla la paleta con las construcciones habilitadas para los diagramas de estructura estática. La Figura E.3 detalla dicha paleta, mediante la cual se accede a las construcciones básicas para el modelado de SMA:

- Agente: Permite definir entidades del tipo agente.
- Recurso: Definir entidades del tipo recurso.
- Ambiente: Permite la definición de la entidad Ambiente.
- Asociación: Especificar relaciones entre las diferentes entidades.

También a través de la paleta se puede acceder a construcciones auxiliares, las cuales tienen como objetivo ayudar al usuario a definir diagramas más ilustrativos.

- Agente_Localizado: Permite definir entidades de Agentes que tienen una ubicación determinada.
- Recurso_Localizado: Permite definir recursos que tienen una ubicación determinada.
- Celda: Representa una ubicación espacial.
- DiagramaClase: Contenedra a las entidades a definir en el modelo

Figura E.3: Paleta del editor estático

Como primer paso para comenzar la definición del diagrama de estructura estática, se debe seleccionar la construcción *DiagramaClase*, en la Figura E.4 se muestra la creación de esta construcción asignándole el nombre *Dinámica Parcelaria*.

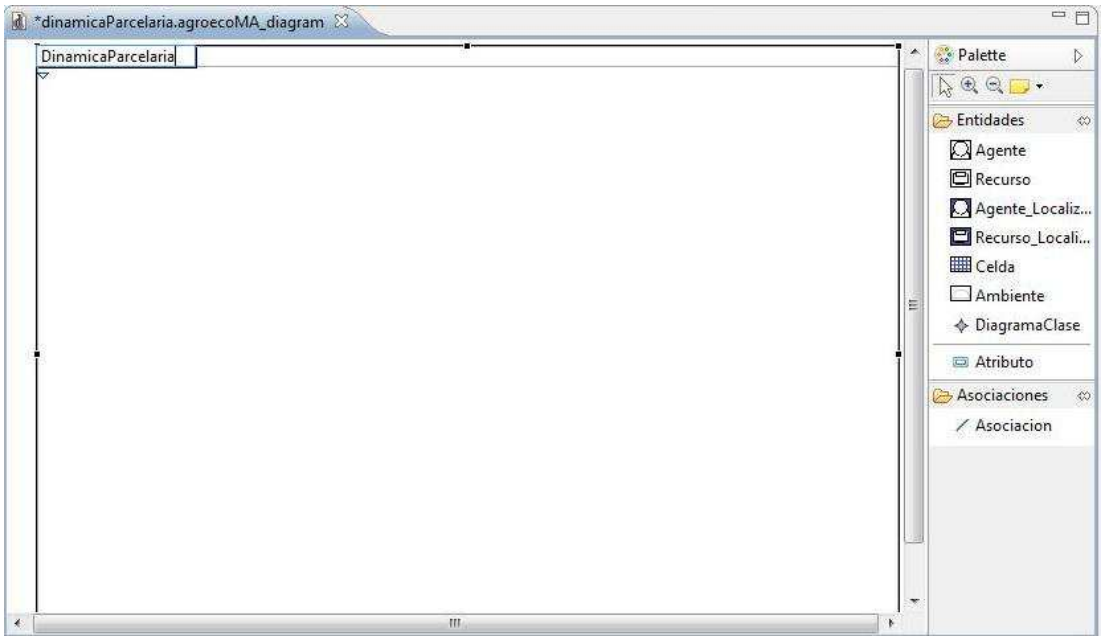

Figura E.4: Ventana para la construcción de diagramas de clase

A continuación se define el diagrama de clase (ver Figura E.5) para el caso de estudio especificado en el capítulo 5.

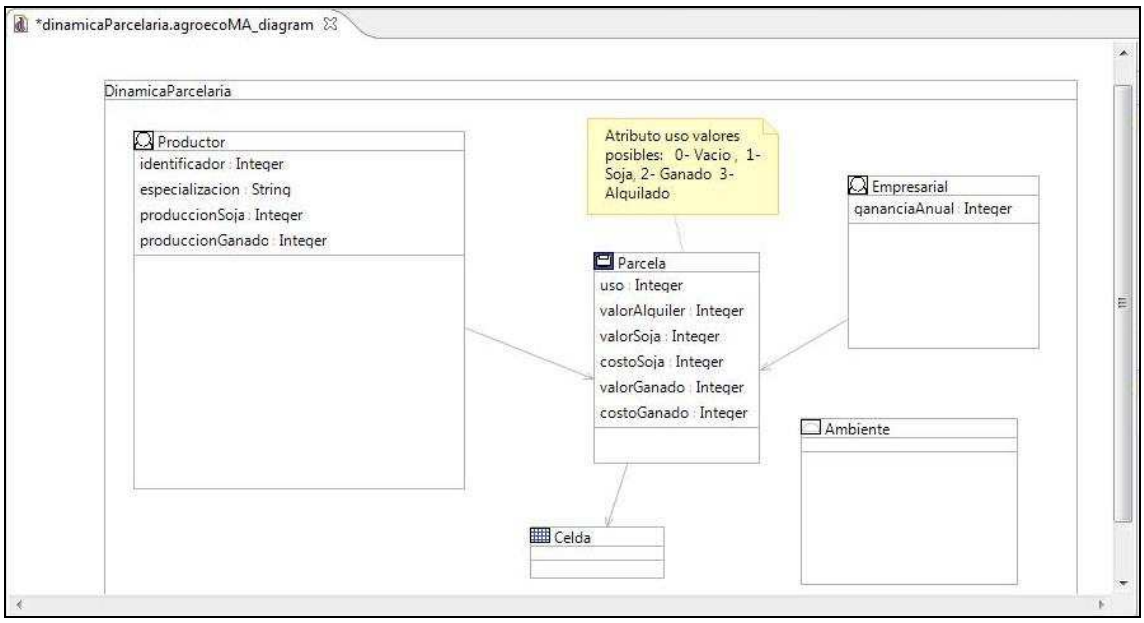

Figura E.5: Diagrama estructura estática realidad Dinámica Parcelaria

FW4SimSMA proporciona la validación de los diagramas en base a restricciones definidas, por ejemplo, cada entidad debe de tener un nombre asignado. En la sección E.3 de este apéndice se ilustra un ejemplo detallado del uso de las validaciones.

E.3. Definición de comportamiento

Para definir el comportamiento de una entidad se debe de hacer click derecho sobre la entidad y seleccionar *Definir comportamiento*, automáticamente se abre el editor de diagramas de comportamiento, mostrando la paleta (ver Figura E.8) con las construcciones disponibles y generando los archivos con las extensiones *.agroecoMAD* y *.agroecoMAD_diagram*.
La Figura E.6 ilustra la forma de acceder a la opción de generación de comportamiento y la Figura E.7 muestra los diagramas generados para la entidad Productor definida para el caso de estudio.

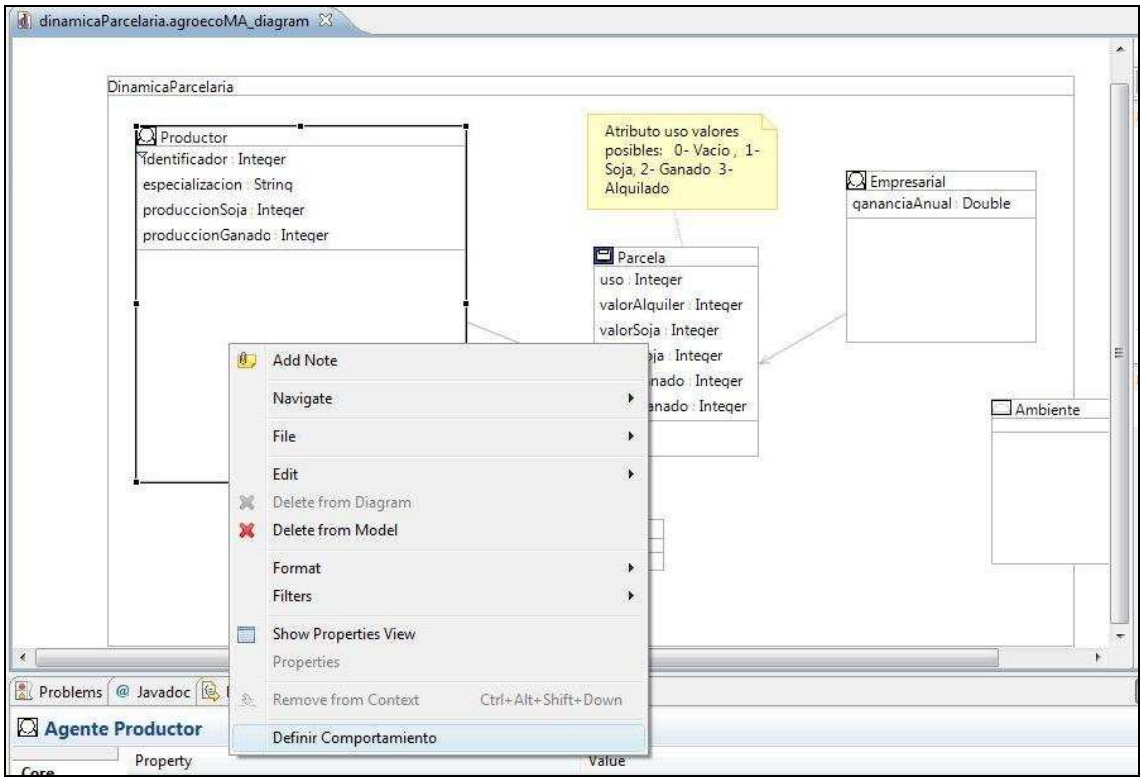

Figura E.6: Acceso a la definición de comportamiento.

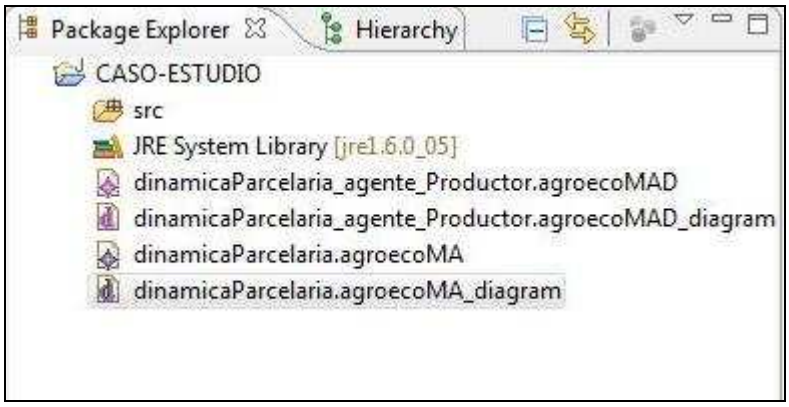

Figura E.7: Archivos de diagrama de comportamiento generados.

Figura E.8: Paleta del editor diagrama comportamiento

Como primer paso para comenzar la definición el comportamiento se debe seleccionar la construcción *DiagramaActividad*, en la Figura E.9 se muestra la creación de esta construcción asignándole el nombre Comportamiento.

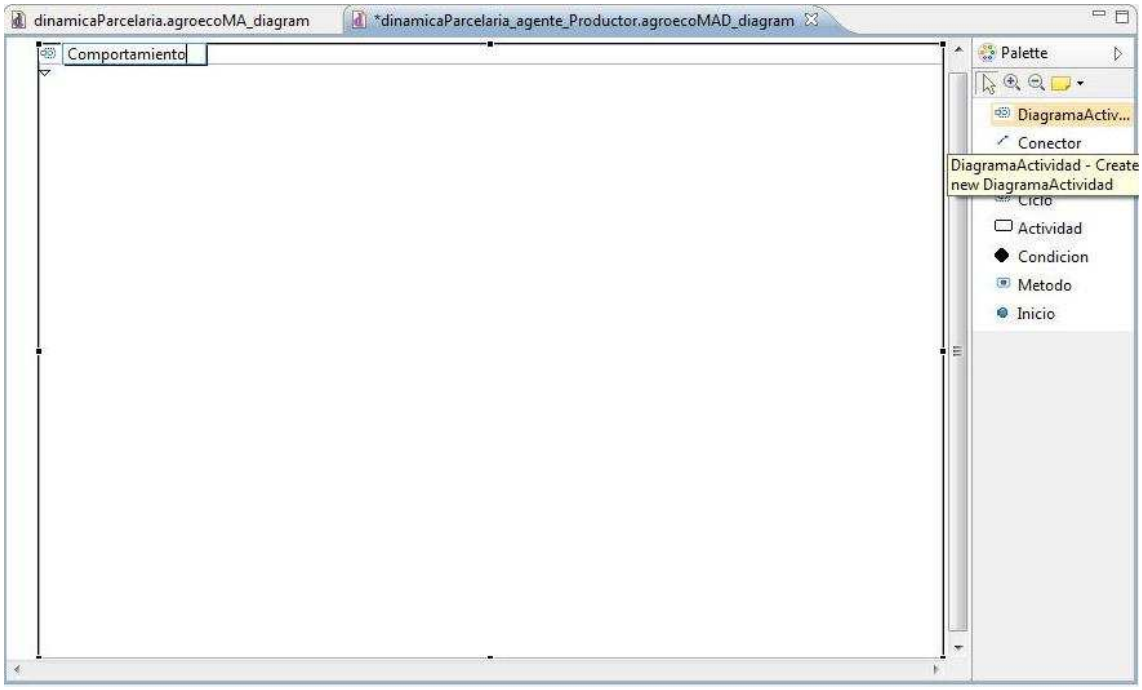

Figura E.9: Ventana para la construcción de diagramas de actividad.

Luego se procede a la definición del comportamiento en base a la realidad a modelar, en la Figura 5.5 del capítulo 5 se ejemplifica el diagrama definido para el agente Productor.

A modo de ejemplo, se dejó la primera actividad sin nombre para evidenciar los controles existentes sobre la definición de los diagramas. Para ejecutar la validación es necesario seleccionar de la barra de herramientas la opción *Diagram/Validate*, la Figura E.10 muestra como acceder a la validación.

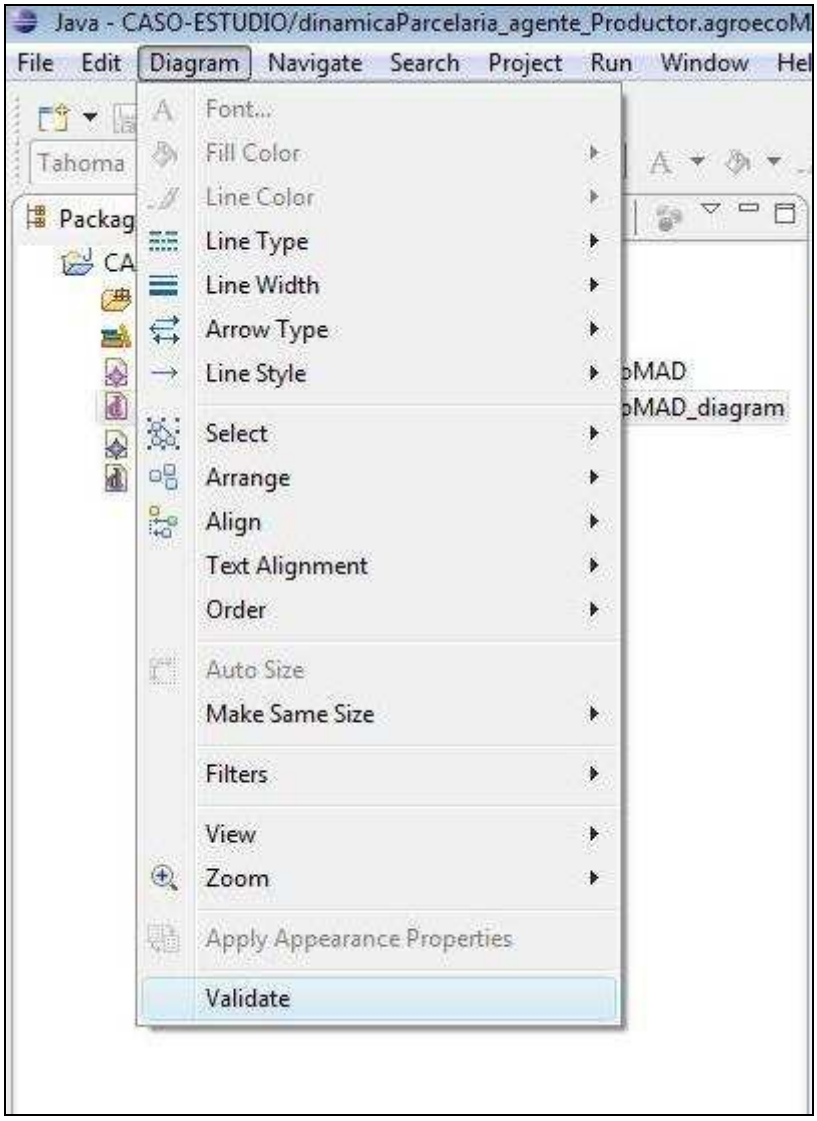

Figura E.10: Acceso a validación.

El error se manifiesta como un círculo rojo sobre la entidad correspondiente que no cumple con las restricciones al modelo, tal como se ilustra en la Figura E.11.

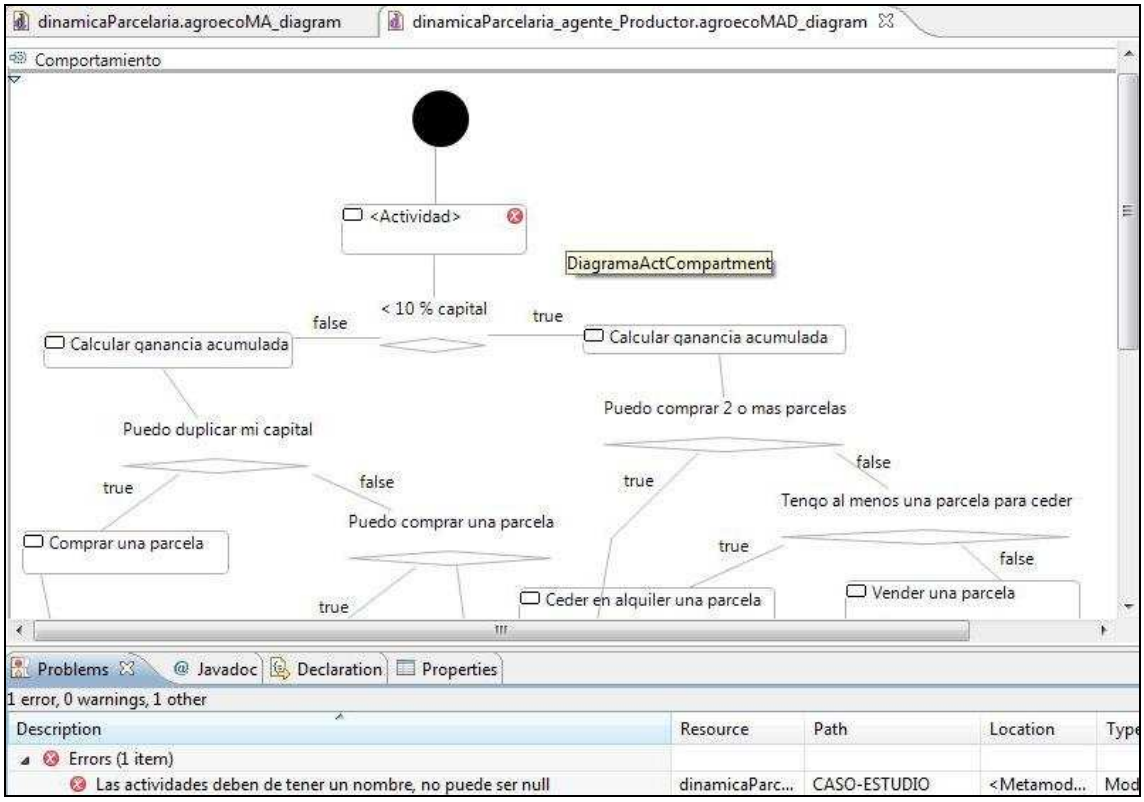

Figura E.11: Ejemplo de validación.

Ingresando el nombre a la Actividad y realizando nuevamente la validación, el error ya no es señalizado por el editor.

E.4. Generación de código

Luego de completar la definición de los comportamientos se realiza la generación de código. Para acceder a la opción de generación se debe de hacer click derecho sobre el diagrama de extensión *.agroecoMA*, seleccionando la opción *Generar Código*, en la Figura E.12 se muestra dicha acción.

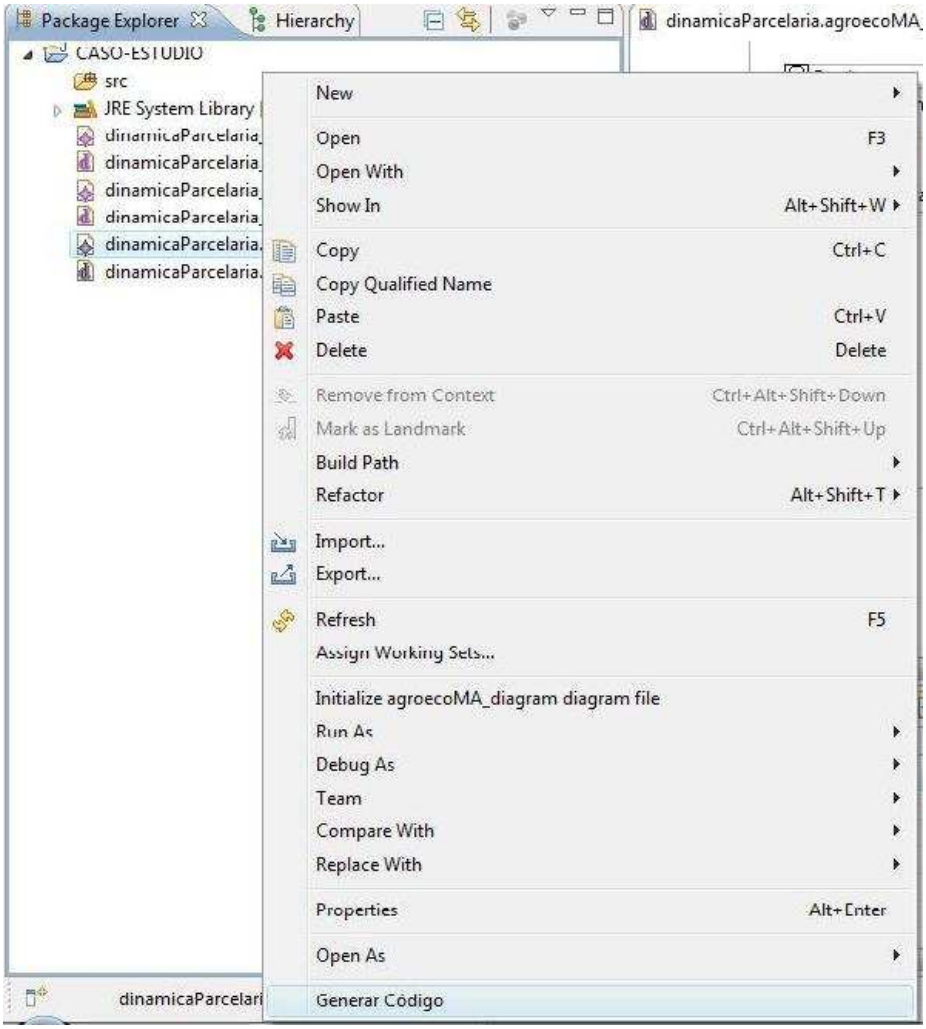

Figura E.12: Acceso a la opción de generación código.

Luego de seleccionar la opción de generación, FW4SimSMA emite un mensaje indicando si la generación fue o no exitosa.

Las clases generadas necesarias para la ejecución en ASCAPE se ilustran en la Figura E.13.

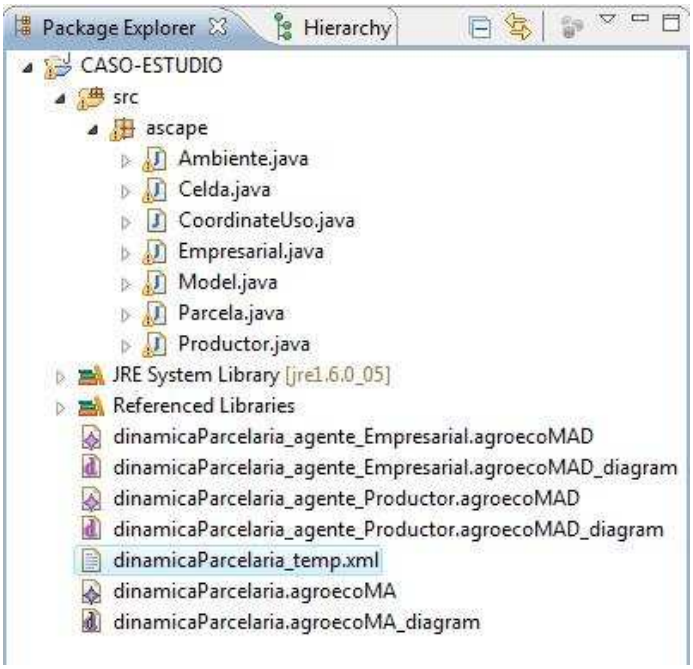

Figura E.13: Clases generadas.

En la Figura E.13 podemos visualizar el archivo dinamicaParcelaria_temp.xml utilizado (como entrada) por el modulo de generación. Este archivo es el resultado de la unión del diagrama de estructura estática (diagrama de clases) con cada uno de los diagramas de comportamientos definidos.

E.5. Ajustes de código

Finalizada la generación el usuario deberá realizar los ajustes y agregados necesarios para la ejecución de la simulación. Quien este encargado de esta tarea, debe contar con conocimientos de Ascape para poder realizar las modificaciones satisfactoriamente.

E.6. Ejecución de la simulación

Para ejecutar la simulación se debe de hacer click derecho sobre el proyecto y seleccionar *Run As/Run Configurations*. Luego se debe definir la clase a ejecutar y sus argumentos, la Figura E.14 muestra la configuración de la clase a ejecutar y la Figura E.15 el argumento necesario.

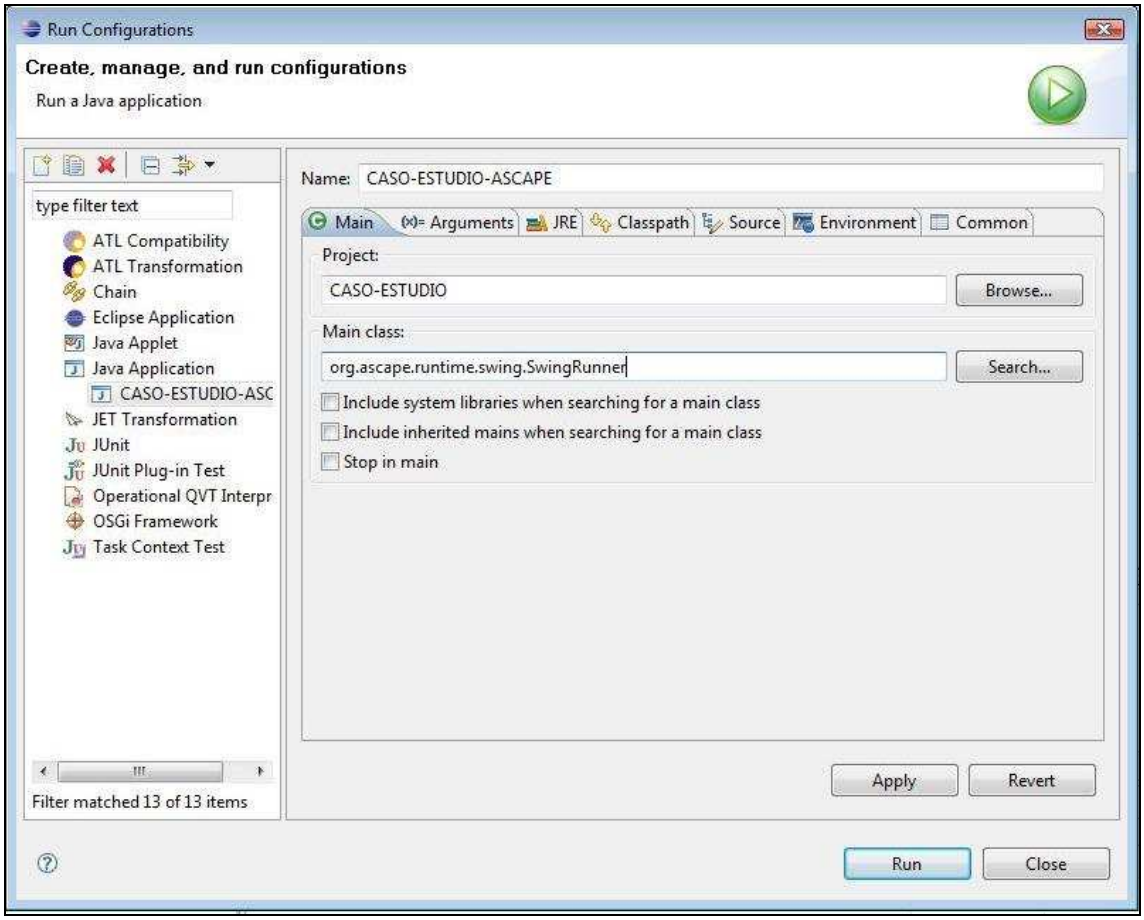

Figura E.14: Configuración clase a ejecutar

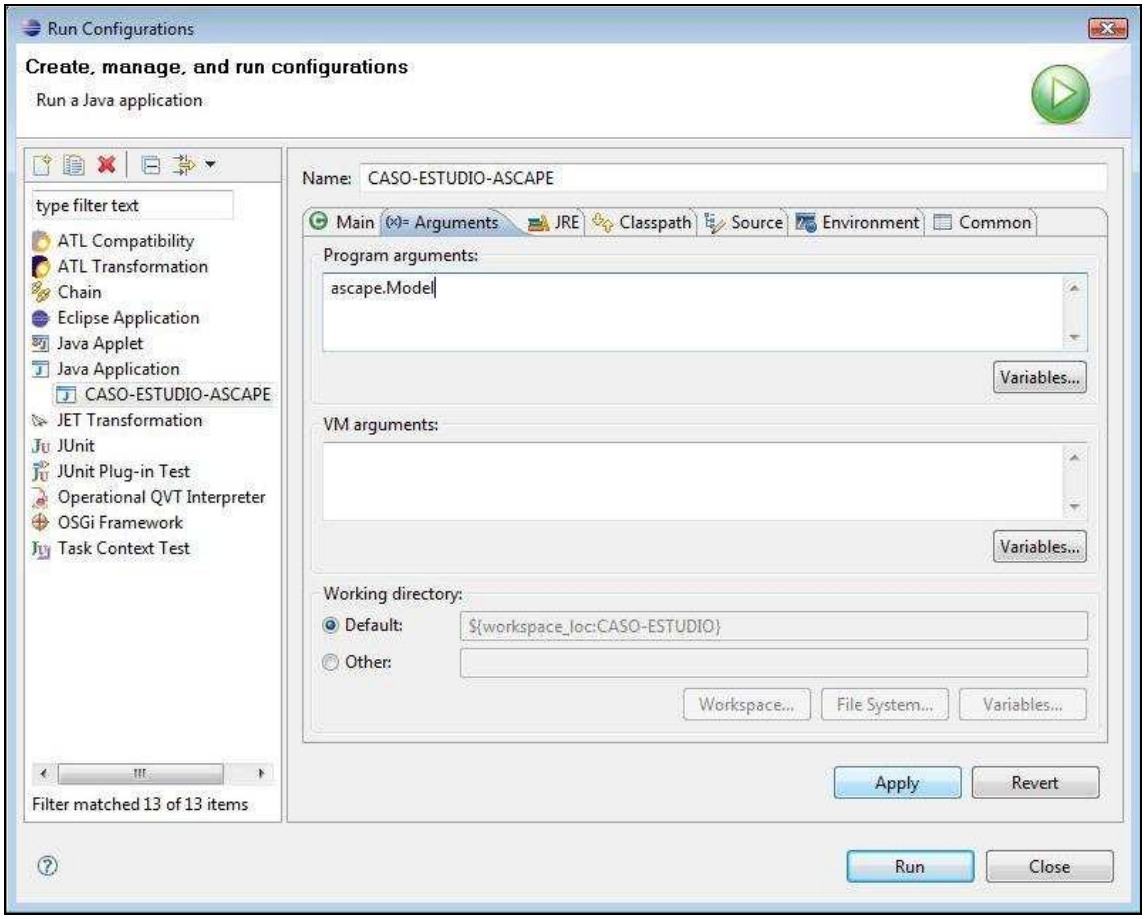

Figura E.15: Configuración de argumentos.

Al ejecutar la simulación, se visualiza (Figura E.16) la interface de usuario provista por Ascape. En la grilla ilustrada se puede observar la interacción en un instante dado entre las entidades modeladas (ver capítulo 5).

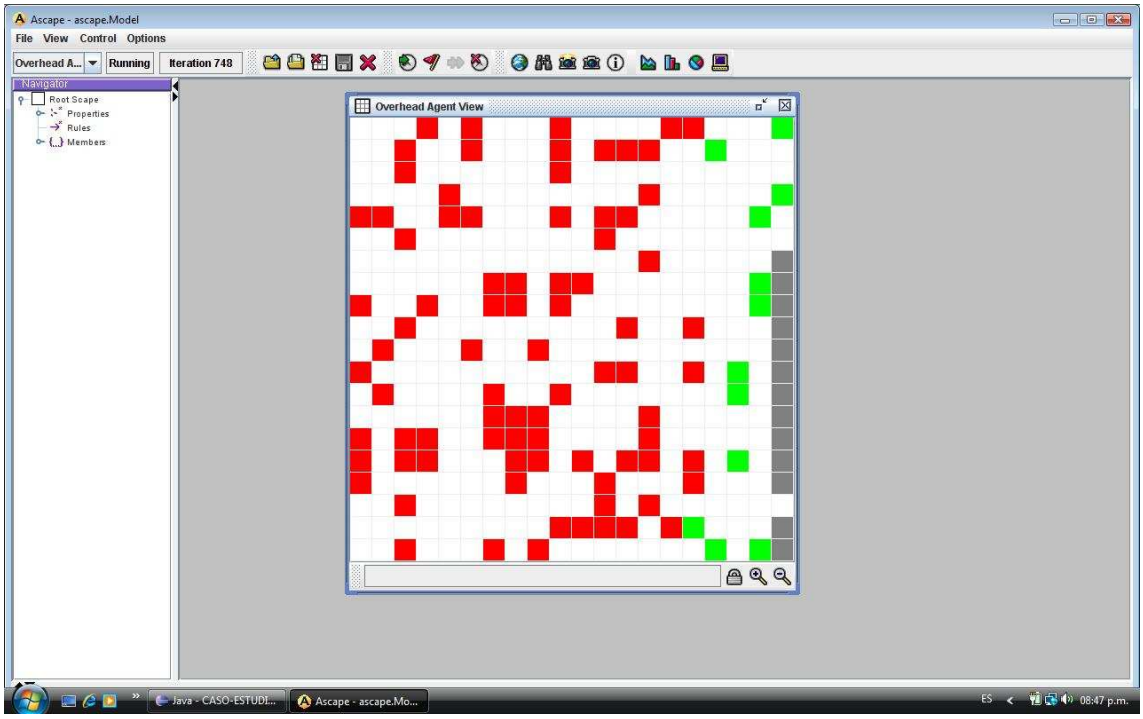

Figura E.16: Ejecución de una simulación.

Apéndice F
Gestión del Proyecto

En este apéndice se detalla la planificación, coordinación y forma de trabajo realizado en el proyecto, detallando las principales tareas llevadas a cabo.

F.1. Descripción

El grupo de trabajo estuvo compuesto por tres estudiantes y la forma de trabajo consistió en la planificación y asignación de un conjunto de tareas, que si bien la mayor parte fueron realizadas en conjunto, existieron algunas que fueron asignadas solo a algunos de los integrantes del grupo.
Un conjunto de tareas fueron planificadas al inicio del proyecto, pero otras fueron definidas en el transcurso del mismo. Mediante reuniones quincenales, conjuntamente con los tutores se planificaron y coordinaron las tareas estimando los plazos para su realización.
Para la organización, almacenamiento y respaldo del código del proyecto y la documentación, se utilizo un repositorio de información el cual podía ser consultado por cualquiera de los integrantes del grupo.
El proyecto tuvo una duración de 15 meses comprendido en el periodo entre marzo del 2009 y junio del 2010. El mismo consto de un conjunto de etapas, donde cada una se compuso a su vez de un conjunto de tareas, las cuales detallamos en las siguientes secciones.

F.2. Estado del Arte

Se basó en la búsqueda de material que permitiera contextualizar y comprender los sistemas Multi-Agente, la simulación basada en SMA, los agro-ecosistemas y los modelos basados en agentes. También comprendió la búsqueda e investigación de los frameworks de simulación basados en SMA existentes en la actualidad.
Las tareas que se desarrollaron en esta etapa fueron:

- Búsqueda en amplitud del material teórico.
- Clasificación y organización de la bibliografía, contextualizando los conceptos básicos enmarcados en el proyecto.
- Establecimiento de criterios para el estudio de los frameworks.
- Búsqueda, categorización y selección de frameworks de simulación basados en SMA.
- Estudio extendido de los frameworks seleccionados.
- Realización de documentación con resultados obtenidos.

F.3. Construcción del framework de simulación basado en SMA(FW4SimSMA)

Luego de realizada la etapa anterior se prosiguió con la construcción de FW4SimSMA, para el cual se realizaron las siguientes tareas:

- Análisis de EMF, GMF y Acceleo para determinar el grado de interoperabilidad y la factibilidad de su uso.
- Definición del modelo.
- Desarrollo del editor gráfico basado en EMF y GMF.
- Desarrollo del generador de código basado en Acceleo.
- Construcción de FW4SimSMA basado en la interoperabilidad de los componentes descriptos en los puntos anteriores.
- Testing de FW4SimSMA.
- Elaboración de documentación para la configuración e instalación de FW4SimSMA.

F.4. Desarrollo del caso de estudio

Con el fin de evaluar FW4SimSMA, se implemento un caso de estudio basado en la simulación de agro-ecosistemas. Las tareas realizadas se detallan a continuación:

- Estudio y planteo de la realidad a modelar.
- Implementación del caso con FW4SimSMA.
- Evaluación de resultados obtenidos.

F.5. Informe Final

Consistió en la realización de un informe en el que se detalla las etapas antes desarrolladas y concluye en base a los resultados obtenidos. Esta comprendió:

- Recopilación y selección del material teórico investigado y documentación realizada en etapas previas.
- Armado de la estructura del informe y generación del contenido.
- Revisiones y correcciones intermedias.

F.6. Cronograma

En la Figura F.1 se puede visualizar la planificación de las tareas realizadas y el período de tiempo dedicado a cada una de ellas.
En la Figura F.2 se ilustra la asignación de recursos realizada por tareas, además se marcan con color naranja aquellas tareas que fueron realizadas en paralelo.

Nombre	Fecha de inicio	Fecha de fin	Duración
Estado del Arte	20/03/09	3/09/09	119
Búsqueda en amplitud de material	20/03/09	24/04/09	25
Clasificación y organización de la bibliografía	24/04/09	5/05/09	7
Definicion de criterios para el estudio de Frameworks	5/05/09	16/05/09	9
Búsqueda y selección de Frameworks	12/05/09	24/06/09	31
Estudio extendido de Frameworks seleccionados	24/06/09	15/08/09	38
Documentación de resultados obtenidos	17/08/09	3/09/09	13
Construcción del Framework de simulacion basado en SMA	3/09/09	29/05/10	192
Análisis de EMF, GMF y Acceleo	3/09/09	20/10/09	33
Definición del modelo de dominio	20/10/09	14/11/09	19
Desarrollo del editor gráfico	16/11/09	20/02/10	70
Editor diagrama de clases	16/11/09	19/12/09	25
Editor diagrama de actividad	4/01/10	2/02/10	21
Realizacion de validaciones a modelos	2/02/10	20/02/10	14
Desarrollo del generador de código	4/01/10	24/02/10	37
Generación del Framework basado en interoperabilidad de componentes	26/02/10	10/03/10	8
Test	15/03/10	7/04/10	17
Documentación para la instalación y configuración	25/05/10	29/05/10	4
Desarrollo del caso de estudio	22/03/10	8/05/10	35
Implementación del caso	2/04/10	1/05/10	21
Evaluación de resultados obtenidos	3/05/10	8/05/10	5
Estudio y planteo de la realidad a modelar	22/03/10	2/04/10	9
Informe Final	6/04/10	14/06/10	49
Recopilación y selección de material	6/04/10	10/04/10	4
Armado de estructura y generación de contenido	12/04/10	1/06/10	36
Revisiones y correcciones intermedias	5/05/10	14/06/10	28

Figura F.1: Planificación y duración de tareas.

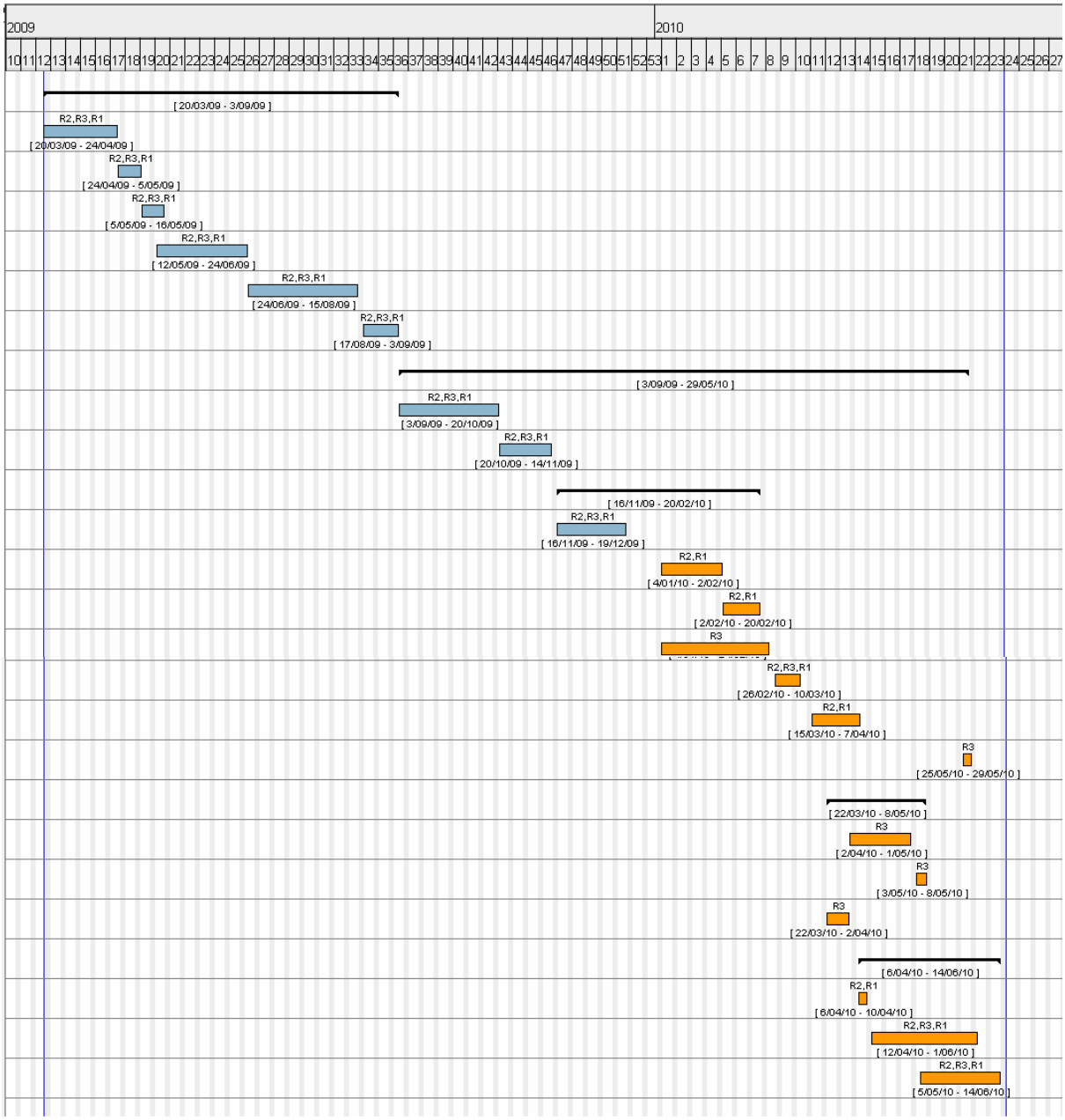

Figura F.2: Asignación de recursos por tarea.

Apéndice G
Especificación técnica

En este apéndice se detallan los pasos necesarios para la instalación y configuración de FW4SimSMA, así como las versiones de las distintas herramientas utilizadas.

G.1. Herramientas

JDK: El Java Development Kit (JDK) utilizado para el desarrollo fue 1.6.0_05-b13.

IDE Eclipse: Como IDE se utilizo Eclipse Ganymede versión 3.4.2 [57], este release incluye los plugins EMF y GMF utilizados en el proceso de desarrollo de los editores gráficos.

Instalación Acceleo: Para instalar el plugin Acceleo se debe utilizar el mecanismo que provee Eclipse para la descarga y actualización de plugins, a continuación se detallan los pasos necesarios.

Como primer paso se debe de ir a la barra de herramientas y seleccionar *Help/Softwate Updates*. A continuación se deberá seleccionar la pestaña *Available Software/Add site*. Luego de realizada dicha acción se despliega un menú en el cual se deberá de ingresar la siguiente URL http://www.acceleo.org/update/ tal como se muestra la en la Figura G.1.

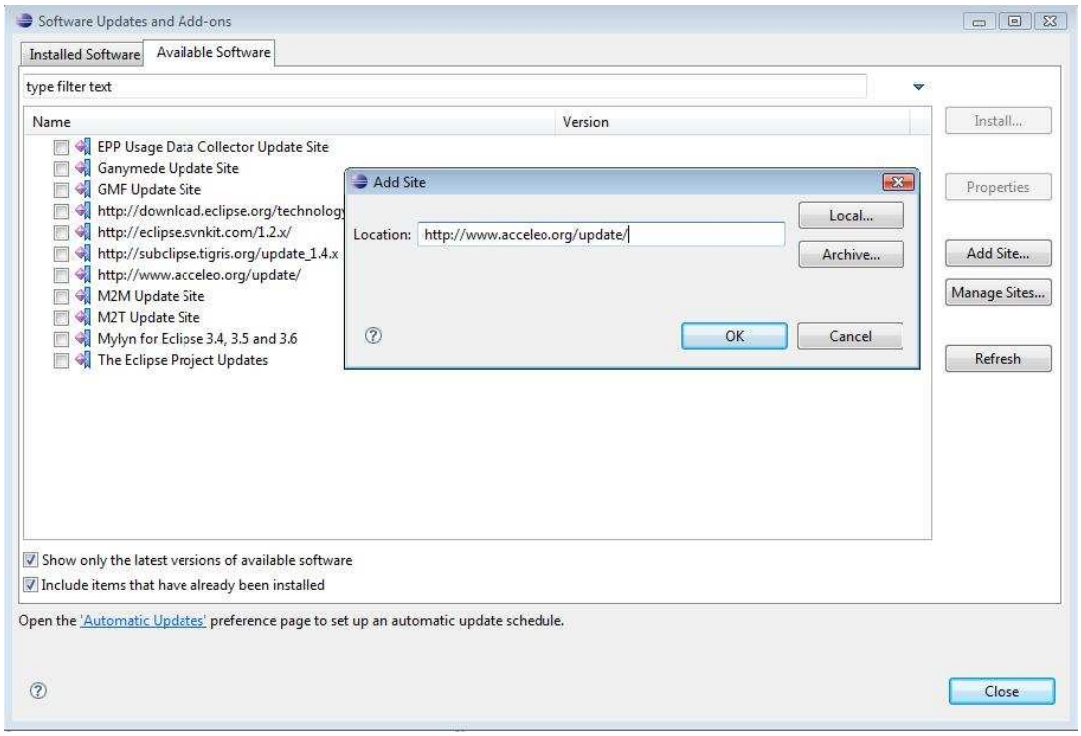

Figura G.1: Instalación de Acceleo.

Por último se selecciona ok para que comience el proceso de descarga e instalación.

Ascape: La versión de Ascape utilizada es la 5.2.0, la biblioteca viene incluida en los jars que componen FW4SimSMA.

G.2. Instalación y descripción de la distribución de FW4SimSMA

La distribución realizada del FW4SimSMA está compuesta por los siguientes jars:

- agroeco.mas.code.gen_1.0.0.201005161719.jar

  Contiene el modulo de generación de código.

- GenModelMA.diagram_1.0.0.201005161719.jar
- GenModelMA.edit_1.0.0.jar
- GenModelMA.editor_1.0.0.jar
- GenModelMA_1.0.0.jar

  Librería para el uso del editor gráfico de los diagramas de estructura estática.

- GenModelMADinamico.diagram_1.0.0.201005161719.jar
- GenModelMADinamico.edit_1.0.0.jar
- GenModelMADinamico.editor_1.0.0.jar
- GenModelMADinamico_1.0.0.jar

  Librería para el uso del editor gráfico de los diagramas de comportamiento.

En la Figura G.2 se ilustra las dependencias que existen entre los distintos componentes que constituyen a FW4SimSMA.

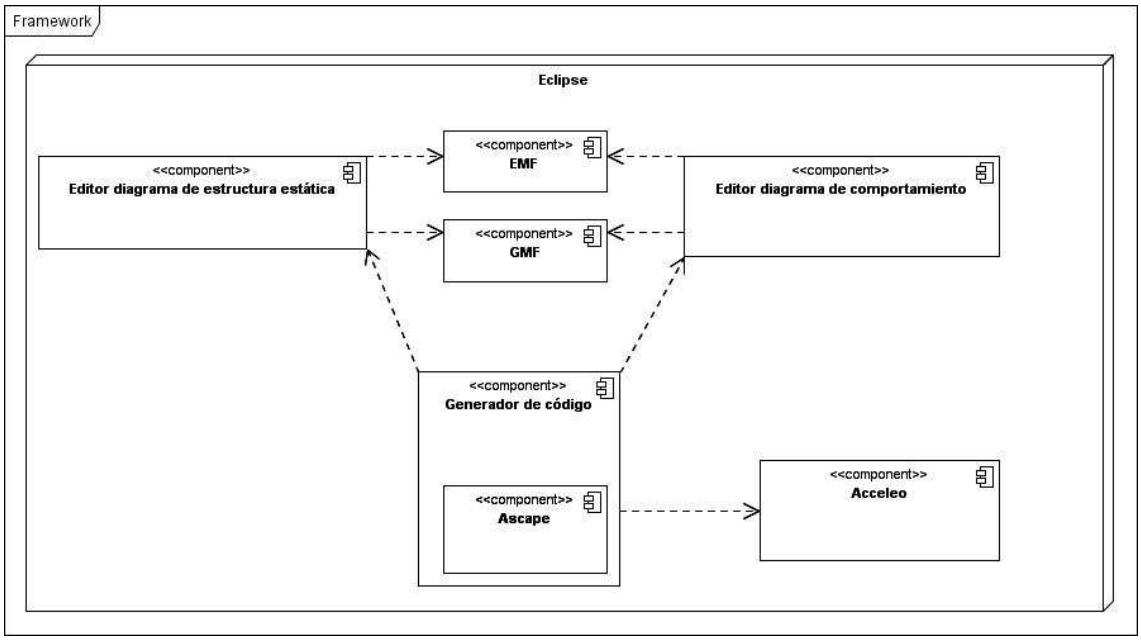

Figura G.2: Componentes desarrollados.

Instalación: La instalación del FW4SimSMA solo es necesario copiar los jars anteriormente mencionados en la carpeta plugins bajo el directorio de instalación de Eclipse, por ejemplo C:\eclipse\plugins.