Universidad Tecnológica Nacional
Facultad Regional Tucumán

Trabajo Final Integrador

Desarrollo de una herramienta CASE para el soporte del modelado organizacional para sistemas multiagentes

Autor:
Ing. Pedro Bernabé Araujo

Director:
Dr. Sebastián Rodríguez

Este Trabajo Final Integrador cumple con los requisitos para el grado de Especialista en Ingeniería en Sistemas de Información

en

Grupo de Investigación en Tecnologías Avanzadas Informáticas
Universidad Tecnológica Nacional - Facultad Regional Tucumán

26 de marzo de 2015

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL TUCUMÁN

Abstract

Especialista en Ingeniería en Sistemas de Información

Desarrollo de una herramienta CASE para el soporte del modelado organizacional para sistemas multiagentes

por Ing. Pedro Bernabé Araujo

El presente Trabajo Final Integrador tiene como finalidad mostrar el desarrollo de la herramienta de modelado Janeiro Studio y sus principales características. Janeiro provee soporte para la modelización de sistemas multiagentes basados en el enfoque organizacional, más precisamente el propuesto en la metodología ASPECS. El desarrollo de la mencionada herramienta requirió de una infraestructura que permita una correcta gestión del proyecto además del uso de un número considerable de frameworks que potencian las características de Janeiro; todo ellos serán presentados a lo largo de este documento.

En la actual fase de desarrollo en la que se encuentra Janeiro, el mismo brinda soporte a la primera fase de desarrollo de la mencionada metodología. Además, cuenta con la posibilidad de realizar validaciones de los modelos y gestión de los elementos de los diagramas. Por último, y a medida que los distintos diagramas que componen la herramienta son presentados, se utiliza como ejemplo un caso de estudio real para demostrar la validez de Janeiro Studio.

Agradecimientos

A mis padres, hermanos y sobrinos que sin su constante apoyo el camino que elegí hubiera sido más difícil y complicado.

A mi director, Dr. Sebastían Rodríguez, por el apoyo, sus consejos e infinita paciencia.

A mis compañeros del GITIA por su colaboración y apoyo.

A mis amigos, por su compañia y buenos deseos en las distintas etapas de mi vida.

Índice general

Abstract I

Agradecimientos II

Lista de Figuras V

Lista de Tablas VI

Abreviaciones VII

1. Introducción 1
- 1.1. Objetivos de este trabajo . . . 2
- 1.2. Plan del documento . . . 2

2. Panorama General de la Ingeniería de Software Orientada a Agentes 5
- 2.1. Introducción . . . 5
- 2.2. Metamodelos . . . 7
- 2.3. Metodologías . . . 8
- 2.4. Herramientas . . . 13
- 2.5. El Metamodelo Organizacional CRIO . . . 17
- 2.6. La Metodología ASPECS . . . 18
- 2.7. Conclusiones . . . 22

3. Entorno de Desarrollo 23
- 3.1. Infraestructura de Desarrollo . . . 23
  - 3.1.1. Sistema de Control de Versión . . . 24
  - 3.1.2. Herramienta de compilación automática y gestión de dependencias 26
  - 3.1.3. Integración Contínua . . . 29
- 3.2. Plataformas . . . 31
  - 3.2.1. Rich Client Platform . . . 31
  - 3.2.2. Modelado . . . 33
  - 3.2.3. Parte Gráfica . . . 35
  - 3.2.4. Integración . . . 36
- 3.3. Conclusiones . . . 37

4. Janeiro Studio **39**
4.1. Introducción . . . 39
4.2. Metamodelos de Requerimientos y de Ontología . . . 42
4.3. Adaptación del Metamodelo CRIO . . . 43
4.3.1. Vista Organizacional . . . 44
4.3.2. Vista de Interacción . . . 45
4.3.3. Vista de Comportamiento . . . 46
4.4. Utilización de Janeiro Studio en ASPECS . . . 47
4.4.1. Caso de Estudio . . . 47
4.4.2. Diagrama de Requerimientos del Dominio . . . 48
4.4.3. Diagrama de Ontología del Problema . . . 48
4.4.4. Diagrama Organizacional . . . 49
4.4.5. Diagrama de Interacción . . . 51
4.4.6. Diagrama de Comportamiento . . . 53
4.5. Conclusiones . . . 54

5. Conclusiones y Trabajos Futuros **55**

Bibliografía **58**

Índice de figuras

2.1. Niveles de abstracción. 8
2.2. Metamodelo CRIO. 18
2.3. Dominios de ASPECS. 19
2.4. Proceso de ASPECS. (figura extraida de [1]) 20

3.1. Arquitectura GIT. 25
3.2. Manejo de versiones en Git. 26
3.3. Ciclo de vida de un proyecto Maven. 27
3.4. Manejo de versiones en GIT. 30
3.5. Ecore en forma de árbol. 34
3.6. Graphical Modeling Framework. 35
3.7. Graphical Modeling Framework. 36

4.1. IDE del prototipo Janeiro Studio. 40
4.2. Metamodelo para el Diagrama de Requerimientos del Dominio. 42
4.3. Metamodelo para el Diagrama de Ontologías del Dominio. 43
4.4. Metamodelo CRIO representado en el Ecore de EMF. 44
4.5. Diagrama de Requerimientos del Dominio 48
4.6. Diagrama de Ontología de Problema 49
4.7. Diagrama Organizacional. 51
4.8. Diagrama de Interacción. 53
4.9. Diagrama de Comportamiento. 54

Índice de cuadros

2.1. Metodologías y sus herramientas 14

4.1. Ecores y sus diagramas 41
4.2. Metamodelo del Dominio del Problema y sus Vistas 44

Abreviaciones

ACL	**A**gent **C**ommunications **L**anguage
ACMAS	**A**gent-**C**entered **M**ultiagent **S**ystems
AGR	**A**gent **G**roup **R**ole
ADELFE	**A**telier de **D**éveloppement de **L**ogiciels 'a **F**onctionnalité **E**mergente
CRIO	**C**apacity **R**ole **I**nteraction **O**rganization
CASE	**C**omputer **A**ided **S**oftware **E**ngineering
CIM	**C**omputation **I**ndependent **M**odel
CVS	**C**oncurrent **V**ersions **S**ystem
DRD	**D**iagrama de **R**equerimientos del **D**ominio
DOD	**D**iagrama de **O**ntología del **D**ominio
DSL	**D**omain **S**pecific **L**anguage
EMF	**E**clipse **M**odeling **F**ramework
EMP	**E**clipse **M**odeling **P**roject
FIPA	**F**oundation for **I**ntelligent **P**hysical **A**gents
GMF	**G**raphical **M**odeling **F**ramework
GEF	**G**raphical **E**diting **F**ramework
IDE	**I**ntegrated **D**evelopment **E**nviroment
IDK	**I**ngenias **D**evelopment **K**it
ISOA	**I**ngeniería de **S**oftware **O**rientada a **A**gentes
JVM	**J**ava **V**irtual **M**achine
KQML	**K**nowledge **Q**uery and **M**anipulation **L**anguage
MDA	**M**odel **D**riven **A**rchitecture
MDD	**M**odel **D**riven **D**evelopment
MVC	**M**odel **V**iew **C**ontroller
OMG	**O**bject **M**anagement **G**roup

OZS	**O**bject-**Z** **S**tatechart
OCMAS	**O**rganizational-**C**entered **M**ultiagent **S**ystems
O-MaSE	**O**rganization-based **M**ulti-agent **S**oftware **E**ngineering
PIM	**P**latform **I**ndependent **M**odel
PSM	**P**latform **S**pecific **M**odel
POM	**P**roject **O**bject **M**anagement
PDT	**P**rometheus **D**esign **T**ool
PTK	**P**ASSI **T**ool**K**it
PASSI	**P**rocess for **A**gent **S**ocieties **S**pecification and **I**mplementation
PDA	**P**ersonal **D**evice **A**ssistant
RCP	**R**ich **C**lient **P**latform
RCA	**R**ich **C**lient **A**pplication
RUP	**R**ational **U**nified **P**rocess
SPEM	**S**oftware **P**rocess **E**ngineering **M**etamodel
SMA	**S**istemas **M**ulti-**a**gentes
UI	**U**ser **I**nterface
UML	**U**nified **M**odeling **L**anguage
XMI	**X**ML **M**etadata **I**nterchange

Dedicado a las personas que creyeron en mi.

Capítulo 1

Introducción

El enfoque multi-agentes propone un marco metodológico bien adaptado para el análisis y modelización de sistemas complejos. Este considera a los sistemas como sociedades compuestas por entidades autónomas e independientes, denominadas agentes, que interactúan con el objetivo de resolver problemas o de realizar conjuntamente una tarea. Los Sistemas Multi-Agentes (SMA) han sido utilizados de forma exitosa en un gran número de dominios que incluyen: robótica, resolución distribuida de problemas, modelización y simulación de sistemas complejos, entre otros[2]. La elaboración de modelos y metodologías adecuadas para este paradigma son de un interés prioritario para una correcta adopción en los medios científicos e industriales. Muchas metodologías, por ejemplo [3–8], consideran las abstracciones organizacionales, tales como organización, rol e interacción, como conceptos centrales que deben ser tomadas en cuenta con el fin de analizar y modelizar un SMA. En efecto, un SMA puede ser descompuesto en múltiples organizaciones en las cuales los agentes juegan roles que corresponden a comportamientos abstractos interactuando entre ellos para satisfacer sus respectivos objetivos. En las últimas tres décadas, investigadores en el campo de los SMA han producido un vasto conjunto de contribuciones para problemas como cooperación, coordinación, negociación, etc. Estas contribuciones apuntan a resolver problemas específicos con diferentes restricciones utilizando técnicas heterogéneas.

Podemos mencionar la utilización de los SMA para el análisis, diseño y modelización de sistemas complejos. Un sistema complejo posee diferentes características siendo una de sus principales aquella que considera al sistema como una composición de diversos elementos que a su vez, cada unos de ellos, pueden ser considerados sistemas complejos. Se han destacado para este enfoque los agentes compuestos por otros agentes[9–12]. En particular podemos resaltar la utilización los sistemas multiagentes holónicos[13]. Estos

consideran que los agentes están compuestos de otros agentes, denominados holones, que permiten la modelización de sistemas jerárquicos.

La Ingeniería de Software Orientada a Agentes (ISOA o AOSE por sus siglas en inglés) requiere para el análisis, diseño e implementación de cuatro elementos fundamentales: el metamodelo y los lenguajes que se utilizarán para describir los modelos; la metodología que define la secuencia de pasos a seguir y los actores involucrados para la obtención de un diseño del producto; la plataforma de implementación sobre la cual se ejecutarán estos modelos; y por último, la herramienta CASE (Computer Aided Software Engineering) utilizada para asistir al diseñador en el proceso de desarrollo. A lo largo del presente Trabajo Final Integrador se detallarán los avances realizados en la herramienta CASE denominada Janeiro Studio para el metamodelo CRIO y la metodología ASPECS.

1.1. Objetivos de este trabajo

Este trabajo tiene como finalidad contribuir en el armado de un ecosistema para la modelización y puesta en funcionamiento de un sistema multiagentes basado en el enfoque organizacional. Denominamos "ecosistema" al conjunto de elementos necesarios para soportar, sistematizar y agilizar el análisis y modelización de un problema partiendo desde los requerimientos iniciales, provistos por los usuarios, hasta la generación de un código ejecutable. El metamodelo, la metodología, la herramienta CASE (Computer Aided Software Engineering) y la plataforma de implementación son los cuatro componentes fundamentales para una correcta modelización de un sistema. A su vez, estas técnicas y/o herramientas están conformadas por otras no menos importantes tales como la validación de modelos, trazabilidad, reglas de transformación, versionado de modelos, etc.

Más específicamente, el trabajo detalla una herramienta CASE para la metodología ASPECS. Esta contribución es dividida en dos: Primero se realizará una presentación de una adaptación de los metamodelos teóricos a una tecnología específica. Segunda, se presenta la creación de los "Editores" y "Vistas" que permiten manipular los metamodelos definidos en esta primera etapa de la herramienta CASE.

1.2. Plan del documento

El presente documento está estructurado de la siguiente manera:

- **Capítulo 2** presenta una revisión de la Ingeniería de Software Orientada a Agentes. En tal revisión se incluye una introducción de las principales abstracciones de los sistemas multiagentes centrados en los conceptos organizacionales (OCMAS, por sus siglas en inglés de Organizational-Centered Multiagent Systems) y sus ventajas. Además se realiza una introducción de lo que significan los metamodelos y su uso, las distintas metodologías que fueron propuesta a lo largo de los años como así también una comparativa de las herramientas de desarrollo disponibles en la actualidad. Por último se realiza una descripción del metamodelo elegido, que es el núcleo principal de la herramienta propuesta, como así también la metodología que será adoptaba por la aplicación en un futuro.

- **Capítulo 3** Divido en dos partes, en este capítulo se presentan las herramientas que fueron necesarias para el desarrollo de la herramienta CASE. La primera parte introduce la infraestructura de desarrollo necesaria para poder llevar a cabo el desarrollo de este tipo de herramienta y que será introducida en el capítulo 4. En la actualidad, no disponer de una infraestructura de desarrollo para la ejecución de un proyecto de cualquier envergadura es considerado un error de concepto y que pueden llevar directo al fracaso. Entre las herramientas más comúnmente utilizadas en los ambientes de desarrollo de software tenemos a los mecanismos de versionado, la herramienta de compilación automática y la de integración de componentes creados por los diferentes desarrolladores, entre otras. En la segunda parte del capítulo se describen las plataformas de desarrollo utilizadas para la adaptación y manipulación de los conceptos organizaciones del metamodelo elegido; en nuestro caso el metamodelo CRIO[14] y la metodología ASPECS[15].

- **Capítulo 4** Se realiza la presentación de la herramienta Janeiro Studio y los motivos que propiciaron su desarrollo. La misma se encuentra en sus primeras etapas de desarrollo (versión beta) por lo que la consideramos aún un prototipo. A lo largo del capítulo se presentan la adaptación del metamodelo CRIO basada en una tecnología específica. Dicha adaptación permite cubrir la primera etapa de la metodología ASPECS que según su especificación está conformada por cinco diagramas que cubren las áreas de requerimientos (representaciones de los requerimientos del dominio y de ontología) como así también los aspectos estructurales, de interacción y de comportamiento. Por último, y para mostrar su validez, se modela un sistema que fue realizado para la empresa PSA-Peugeot.

- **Capítulo 5** se presentan las conclusiones de la Trabajo Final Integrador y cuales serán los trabajos futuros. Se detalla sintéticamente las funcionalidades actuales de Janeiro Studio. Además se introducen los trabajos en curso como así también se plantean las diferentes opciones que se presentan a partir de la creación de

Janeiro. Entre los tópicos que se pueden mencionar están el desarrollo del segundo dominio de ASPECS, el módulo de validación tanto sintáctica como semántica y la identificación de oportunidades en aplicación de patrones de diseño, entre otros.

Capítulo 2

Panorama General de la Ingeniería de Software Orientada a Agentes

2.1. Introducción

En los 80's los Sistemas Multiagentes emergieron como una de las tecnologías más interesantes para abordar sistemas complejos, distribuidos y abiertos. En este sentido los sistemas basados en agentes han demostrado exitosamente su potencial abordando una amplia variedad de problemas que van desde la robótica, resolución de problemas distribuidos, modelado y simulación, sólo por mencionar algunas áreas[2]. El creciente interés se debe, entre otros, al concepto de agente como entidad autónoma. Un agente es una entidad física o virtual con un alto grado de autonomía, independiente, capaz de cooperar, competir, comunicarse, actuar flexiblemente y ejercer control sobre su comportamiento dentro del marco de sus objetivos. Estas entidades evolucionan dentro de un ambiente al cual son capaces de percibir y modificar. Así, un sistema multiagentes está compuesto por múltiples agentes inteligentes que interactúan entre sí para alcanzar objetivos que pueden ser compartidos o no entre los agentes[16].

Diferentes enfoques han sido propuestos para los sistemas multiagentes, entre ellos, el enfoque organizacional ha ganado especial importancia. Inspirado en la metáfora social, es usada tanto en metodologías (GAIA[17], MESSAGE[18], ASPECS[1]) como metamodelos (AGR[19], MOCA[20], CRIO[11]). Este novedoso enfoque es una evolución en la forma de modelar que parte de una visión donde el sistema estaba centrado principalmente en el agente y sus aspectos individuales, hacia una visión en donde el sistema es considerado como una organización en la cual los agentes forman grupos y jerarquías,

además de seguir reglas y comportamientos específicos. Conceptos como "Organización", "Grupo","Comunidad", "Roles", "Protocolos" son términos comunes en este tipo de teoría.

El enfoque antes mencionado permite descomponer los problemas en partes más pequeñas además de proveer el contexto de interacción entre los agentes de cada una de estas unidades. Ferber[21] define que dos niveles son posibles; nivel "organizacional" y nivel "agente". El nivel organizacional o social (el "que") es donde se pueden observar los aspectos dinámicos y estructurales de una organización SMA. Este nivel describe las relaciónes y los patrones de actividad que deberían ocurrir en el nivel de agentes. También, es común encontrar conceptos tales como rol, grupos, comunidades, tareas e interacción. El nivel de agentes (el "como") describe el comportamiento propio del agente. En otras palabras, se detalla la arquitectura interna del agente, sus estados mentales, creencias, deseos, intenciones, metas, y si es reactivo o intencional.

Además, adoptar el enfoque organizacional permite al diseñador tratar a los problemas a través de dos estrategias posibles: la descomposición vertical y la horizontal. La descomposición vertical permite que el comportamiento que representa a la organización sea fragmentado en un conjunto de sub-organizaciones refinando aspectos de la misma. En cambio, la descomposición horizontal, modela las interacciones entre las entidades presentes en el mismo nivel de abstracción el cual es necesario para alcanzar los objetivos requeridos[21].

Como indica Ferber en [21], el enfoque organizacional contribuye a la Ingeniería de Sotware Orientada a Agentes en los siguientes puntos:

- **Heterogeneidad en los lenguajes.** Si cada grupo es considerado como un espacio de interacción, dentro del mismo, podemos encontrar recursos específicos para la comunicación tales como KQML[1] o ACL[2] sin modificar la arquitectura de todo el sistema.
- **Modularidad.** Las organizaciones pueden ser vistas como módulos que proveen una descripción, a partir de la cual se puede obtener un comportamiento particular de los miembros. Usamos el enfoque para definir reglas de visibilidad claras que ayudan en el diseño de un sistema multiagentes.
- **Múltiples arquitecturas.** El enfoque organizacional no hace ninguna suposición sobre la arquitectura interna de los agentes, dejando su especificación abierta a un considerable número de modelos e implementaciones diferentes.

[1] Knowledge Query and Manipulation Language
[2] Agent Communication Language

- **Seguridad de las aplicaciones** Si todos los agentes se comunican sin ningún tipo de control externo, esto puede llevar a problemas de seguridad. Ahora, si permitimos que cada grupo controle el acceso a los roles definidos dentro del mismo, se puede alcanzar un nivel de seguridad sin la necesidad de un control global centralizado.

Considerando a las organizaciones como moldes que pueden ser usados para definir una solución a un problema, creemos que el enfoque organizacional fomenta modelos reusables tanto en un mismo proyecto como en desarrollos futuros.

2.2. Metamodelos

A medida que los sistemas crecen en complejidad, es recomendable el uso de modelos para lograr un mejor entendimiento de la arquitectura y la dinámica de los mismos. En la Ingeniería Dirigida por Modelos (Model Driven Engineering, MDE), iniciativa de la Object Management Group (OMG)[3], se considera al metamodelado como un elemento fundacional donde el lenguaje utilizado para describir metamodelos es el estándar Meta-Object Facility (MOF)[4]. El metamodelado es una actividad muy cercana al modelado que produce, entre otras cosas y valiendo la redundancia, metamodelos.

Un metamodelo formaliza los conceptos que se van a utilizar para construir modelos. En otras palabras, especifica una sintaxis abstracta que define un conjunto de conceptos, su significado, los atributos que lo conforman y cómo estos se relacionan entre sí, como así también las reglas que surgen del metamodelo y sirven para combinar conceptos que permiten construir un modelo parcial o completo [22].

La conveniencia de que los diseñadores basen su trabajo en el empleo de un lenguaje con cierto nivel de abstracción que manejan conceptos más cercanos al dominio de aplicación permite la conformación de los denominados Lenguaje Específico del Dominio (DSL, Domian Specific Language). Existe en la actualidad una creciente tendencia a utilizar metamodelos como base para las metodologías. Allí, el metamodelo provee información acerca de todos los conceptos del dominio que el diseñador puede utilizar en su trabajo y las relaciones que pueden darte entre ellos.

[3] http://www.omg.org
[4] http://www.omg.org/mof

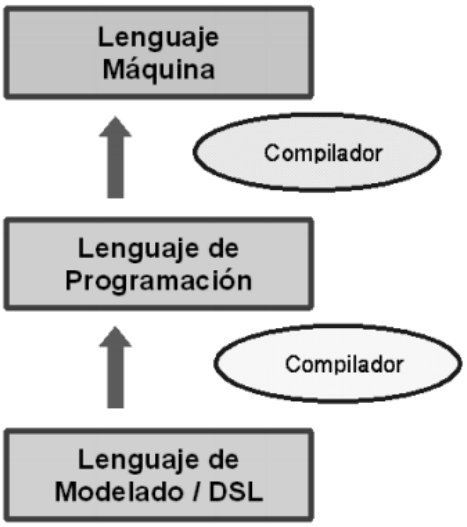

FIGURA 2.1: Niveles de abstracción.

2.3. Metodologías

Una metodología es una secuencia sistematizada de actividades para la obtención de un producto. Brindan un marco de trabajo para estructurar, planificar y controlar el proceso de desarrollo de software.

No fue sino hasta la crisis del software a principios de los 60' en donde comenzaron a aparecer las primeras metodologías. Dos son los motivos que provocaron la crisis, la primera de ellas se debió a que hasta ese entonces el sector informático en su totalidad era bastante inmaduro dado que estaba ligado a las cuestiones relacionadas al hardware sumado a la falta de estándares y/o métodos. El segundo motivo, se debió a los constantes cambios que se fueron dando en la industria gracias a la creciente demanda de software además del incremento de funcionalidades y/o prestaciones de esos software.

A lo largo de los últimos 40 años el desarrollo de metodologías fue uno de los campos más prolíficos en la ingeniería de software en general. Sobre todo en el paradigma Orientado a Objetos que ha alcanzado un gran nivel de madurez debido, entre otras razones, a los años de experiencia acumulados tanto dentro del campo académico como industrial (RUP es la metodología de facto en la actualidad). A su vez, la teoría agentes representa una tecnología relativamente nueva, útil para abordar sistemas complejos por lo que las metodologías propuestas son también más recientes pero diversas y de gran crecimiento. Esta diversidad es probablemente atribuida a que no existen definiciones acabadas de los diversos conceptos que conforman la teoría.

A continuación se describirán las metodologías orientadas a agentes basadas en el enfoque organizacional más importantes dentro de la literatura, destacando en cada una sus principales características.

O-MaSE

Dentro de las metodologías OCMAS, Organization-based Multiagent Systems Engineering (O-MaSE)[23] representa un novedoso enfoque para el análisis de sistemas basados en los conceptos de agentes. Los autores proponen para esta versión de la metodología(la misma es una extensión de MaSE[24]) la posibilidad de personalizar las actividades según las necesidades del proyecto y de la experiencia de los diseñadores.

Fue pensada desde sus comienzos como un conjunto de fragmentos que pueden ser ensamblados por los desarrolladores para así poder capturar los requerimientos específicos de sus proyectos. Con este principio, O-MaSE se aleja de lo establecido por el estándar SPEM[5] de la OMG[6] que propone organizar en fases las distintas actividades de desarrollo que conformarán la metodología. En cambio, O-MaSE, define actividades y tareas y permite a aquellas organizaciones que adopten la metodología organizar dichas actividades de la manera que crean conveniente basados en las necesidades específicas del proyecto.

En rasgos generales, O-MaSE tiene tres estructuras principales:

- Un Metamodelo. En el mismo se definen los conceptos principales, sus atributos y las relaciones entre los conceptos que son necesarios para el diseño e implementación de un SMA.
- Un conjunto de "Method Fragments" que representan las tareas que son ejecutadas para un conjunto de "Work Products" el cual puede incluir modelos, documentos o código del proyecto.
- Conjunto de guías. que define como los "Method Fragments" están relacionados entre sí.

Si bien O-MaSE es una metodología adaptable, es posible describirla en tres fases: Análisis de Requerimientos, Diseño e Implementación. No existen reglas fijas y/o estrictas acerca de que actividades pueden ser incorporadas en cada fase haciendo que esta metodología sea altamente flexible.

[5]Software & Systems Process Engineering Metamodel
[6]Object Management Group, http://www.omg.org

GAIA

Propuesta inicialmente por Woolddridge[25], GAIA representa una de las metodologías más antiguas existentes para el desarrollo de MAS. Con GAIA es posible modelar tanto los aspectos macro (Social) como los micro (interno a los agentes).

Originalmente pensado para el modelado de un SMA cerrado, el proceso estaba compuesto de sólo dos fases: Análisis y Diseño. Un tiempo después, en la versión más actualizada de GAIA, se realizarían ciertas mejoras entre las que podemos destacar: (i) la incorporación de las abstracciones propias del enfoque organizacional. (ii) la reformulación del proceso de GAIA introdujo una nueva fase denominada *Diseño Arquitectónico*. (iii) Se introdujo un conjunto de nuevas clases de modelos para las distintas fases[26].

Ingenias

Ingenias, propuesto por [27], representa una evolución de MESSAGE[28] pero con un refinamiento más completo y consistente de los metamodelos. El motivo de Ingenias fue que los autores consideraban que el campo de los SMA estaba fuertemente fragmentado por diferentes propuestas sobre como aplicar agentes y las perspectivas de los conceptos de agentes en sí mismo. En este contexto, Ingenias emerge como un enfoque de integración capaz de soportar el uso simultáneo de diferentes trabajos. Por este motivo, que considera a los SMA desde cinco puntos de vista: organización (punto de vista central), agente, tareas/metas, interacciones y ambiente. Cada una de estas vistas proveen un conjunto de conceptos y relaciones entre ellas. También considera a la consistencia entre los diferentes puntos de vista como una de sus características centrales.

Prometheus

Prometheus es una metodología propuesta por [29]. Los autores desearon que sea una metodología práctica para la especificación y diseño de un sistema agentes o sistemas multiagentes con soporte de estructuras jerárquicas de diseño que permite modelar múltiples niveles de abstracción. Todo proceso definido en Prometheus es detallado, dado que provee guías de principio a fin, sobre como ejecutar los distintos pasos y/o actividades; desde los requisitos iniciales hasta la generación de código sin olvidar el debugging y las etapas de testing. La metodología está compuesta por tres fases: Especificación de sistema, Diseño de arquitectura y Diseño detallado. Cada una de estas fases incluye modelos que se enfocan en los aspectos estructurales del sistema, los aspectos dinámicos y un formulario de texto que sirve para el detalle de las entidades individuales.

Esta metodología fue usada tanto en los ámbitos académicos como en la industria. Estas experiencias permitieron que la metodología sea estructurada de una forma que facilitó el desarrollo de diversas herramientas de soporte, una de las cuales será presentada en la siguiente subsección.

ROADMAP

En esta misma sección se presentó GAIA y sus características más importante. Sin embargo dicha metodología presentaba debilidades tales como la imposibilidad de abordar sistemas abiertos, representar jerarquías de roles, explicitar la representación de las estructuras sociales y sus relaciones o la poca flexibilidad de incorporar cambios dinámicos. Role Oriented Analysis and Design for Multiagent Programming o mejor conocido por su acrónimo ROADMAP[30] fue concebida reutilizando muchos de las definiciones de GAIA pero incorporando mejoras que permiten superar las "carencias" mencionadas.

TROPOS

Es una metodología propuesta por [31], la misma describe un proceso de desarrollo estructurado que permite modelar los sistemas socio-técnicos. Adopta fuertemente un enfoque dirigido por requerimientos reconociendo la importancia del modelado del dominio de los "Stakeholders" (actores sociales) y el análisis de sus metas estratégicas e interdependencias además de los aspectos organizacionales.

TROPOS está basado en dos ideas principales. La primera, la noción de agente y todas las nociones mentales relacionadas (por ejemplo metas y planes). Conceptos que son usados en todas las fases del desarrollo de software, desde el análisis temprano hasta la implementación. Segundo, Tropos cubre también la fase del análisis *muy temprano* de requerimientos permitiendo así un profundo entendimiento del ambiente donde el software debe operar y el tipo de interacciones que deben ocurrir entre el software y las personas.

Desde sus inicios y hasta la fecha, Tropos ha estado en constante evolución recibiendo actualizaciones que le permiten, por ejemplo, la posibilidad de abordar aspectos de los sistemas distribuidos entre otras características. Actualmente el procesos de TROPOS está compuesto por cinco fases[32]: Análisis de Requerimientos Tempranos, Análisis de Requerimientos Tardíos, Diseño Arquitectónicos, Diseño Detallado, Implementación y Prueba.

Es considerada como una de las metodologías más consolidadas de la literatura de agentes. Esto se debe en parte a que continuamente se le han introducido mejoras sumado a los diversos estudios de casos realizados.

PASSI

El Proceso para la Especificación e Implementación de Sociedades de Agentes (PASSI por sus siglas en inglés) [33] es una metodología paso a paso que abarca desde los requerimientos hasta el código para el diseño y el desarrollo de sociedades multiagentes. Desde su creación PASSI logra integrar modelos y conceptos que provienen de la ingeniería orientada a objetos y de la inteligencia artificial basados en la notación de UML. A su vez, PASSI, fue diseñado para ofrecer a los ingenieros la posibilidad de construir sistemas basados en agentes que cumplan con las exigencias de los estándares propuestos por FIPA[7]. De acuerdo a las especificaciones de FIPA, los agentes son considerados móviles y pueden interactuar mediante la comunicación semántica, que está referida a una ontología y a un protocolo de interacción. Es por esta característica y otras que hacen de PASSI ideal para el modelado de sistemas de mediano/gran tamaño distribuidos en diversos nodos de ejecución.

El proceso de desarrollo está compuesto por cinco modelos: Requerimientos de Sistemas, Sociedades de Agentes, Implementación de Agentes, Código, Implementación. A su vez, cada una de las fases mencionadas pueden ser descompuestas en una o mas subfases. Cada una de estas actividades es responsable o del diseño o el refinamiento de uno o más artefactos que son parte componente del modelo.

ADELFE

ADELFE[34] es el acrónimo en francés de Atelier de Développement de Logiciels à Fonctionnalité Emergente que significa Taller para el Desarrollo de Software con Funcionalidades Emergentes. La metodología fue diseñada para abordar los Sistemas Multiagentes Adaptativos. Este enfoque permite diseñar y construir sistemas multiagentes auto-organizados en donde los agentes solo persiguen un objetivo local mientras mantienen relaciones de cooperación con sus agentes vecinos.

El proceso definido para ADELFE está compuesto de cinco etapas a saber: Requerimientos preliminares, Requerimientos Finales, Análisis, Diseño e Implementación. A su vez estas etapas se descomponen en 21 actividades que producen o refinan alrededor de 12 Work Products.

[7] Foundation for Intelligent Physical Agents, http://www.fipa.org

Desde su publicación en el año 2000 hasta la actualidad, ADELFE cuenta en su registro con al menos 20 aplicaciones significativas repartidas entre proyectos académicos e industriales.

Esta revisión no pretende ser un listado exhaustivo de todas las metodologías propuestas para la tecnología de sistemas multiagentes, sino se enfoca en aquellas que son las más relevantes para la modelización organizacional. La mayoría de las metodologías mencionadas adoptan un proceso de desarrollo iterativo e incremental tal como lo hicieron las metodologías propuestas para el Paradigma Orientado Objetos. Las mismas se pueden clasificar en dos tipos claramente diferenciables: aquellas que definen un proceso formal tales como PASSI, Ingenias, Adelfe o uno informal como GAIA y Tropos. Sin embargo, las etapas del ciclo de vida de desarrollo que abarcan son variadas, por ejemplo GAIA y TROPOS solo cubren la etapas de análisis y diseño mientras que Adelfe, Ingenias y PASSI cubren todas las etapas del proceso de desarrollo.

A su vez, es importante destacar que la mayoría de las metodologías en la actualidad son del tipo independiente del domino con excepción de Adelfe que fue concebida para la modelización de sistemas multiagentes adaptativos.

En la subseccion 2.6 se realiza una descripción detallada de la metodología ASPECS, la cual es el punto de partida del presente documento. La misma es considerada como una de las metodologías más completa existente en la actualidad[35].

2.4. Herramientas

La Ingeniería de Software Orientada a Agentes (ISOA o AOSE por sus siglas en inglés) requiere para el análisis, diseño e implementación de cuatro elementos fundamentales: el metamodelo y los lenguajes que se utilizarán para describir los modelos; la metodología que define la secuencia de pasos a seguir y los actores involucrados para la obtención de un diseño del producto; la plataforma de implementación sobre la cual se ejecutarán estos modelos; y por último, la herramienta CASE (Computer Aided Software Engineering) utilizada para asistir al diseñador en el proceso de desarrollo. Consideramos que estos elementos representan lo que denominados *Ecosistema* y que son necesarios para abordar un proyecto de desarrollo.

En la literatura orientada a agentes existe un importante número de herramientas de desarrollo. La lista está comprendida por aproximadamente 24 aplicaciones comerciales y 40 proyectos académicos. Esta cantidad, y a diferencia del paradigma orientado a

objetos que alcanzó un nivel de madurez considerable, refleja la constante evolución de la teoría agentes. El propósito de estos desarrollos es, en la mayoría de las veces, reforzar las propuestas de algún grupo de investigación ya sea una metodología en particular, teoría de agencia, arquitectura o algún lenguaje agentes.

Dada la vasta diversidad de metáforas utilizadas para el modelado de sistemas multiagentes, realizar una comparativa de todas ellas está fuera del alcance de este documento. Es por ello que la tabla 2.1 sólo se centra en las herramientas que están basadas en el enfoque organizacional. Muchas de estas aplicaciones están desarrolladas para metamodelos diferentes, proveyendo soporte visual para algunos o la mayoría de sus diagramas más importantes.

CUADRO 2.1: Metodologías y sus herramientas

	Metodología	Diagramas Cubiertos	Verificación de Modelos	Verificación Cruzada	Generación de Código	Soporte
agentTool III	O-MaSE	Todos	Si	si	Si	Si
GAIA4E	GAIA	Todos	N/E	N/E	No	N/S
IDK	Ingenias	Principales	No	No	Si	Si
PDT	Prometheus	N/E	Si	Si	Si	?
Metameth / PTK	PASSI	Todos	Si	Si	Si	No/Si
OpenTool	Adelfe	?	Si[8]	No	?	No
Rebel	ROADMAP	Principales	No	No	No	No
T-Tool	Tropos	Principales	Si[9]	No	No	No

La tabla 2.1 consta de una serie de criterios que describiremos brevemente a continuación.

Metodología. Indica cual es el proceso de desarrollo de software a la que brinda soporte.

Diagramas Cubiertos. Cada metodología especifica una serie de diagrama. Este ítem indica en que medida estos diagramas cubren las distintas fases que componen la metodología.

Verificación de Modelos. Permite encontrar inconsistencias en un mismo diagrama.

Verificación Cruzada. Permite validar la consistencia del significado de un mismo concepto en diferentes diagramas que conforman el modelo.

Generación de Código. Contemplamos este criterio dado que deja entrever si la herramienta acompaña el proceso de desarrollo. Es importante remarcar que ninguna de ellas genera un código que directamente es compilable sino más bien nos proporcionan un esqueleto del código.

Soporte. Indica si se continua con el desarrollo de la herramienta. Muchas de ellas fueron pioneras en en área de las tecnologías multiagentes. Otras en cambio fueron creadas para demostrar la efectividad de algún concepto y nunca salieron del ámbito académico. Distinta suerte tuvieron otras como OMASE que cuentan en su haber aplicaciones académicas e industriales.

A continuación describiremos brevemente algunas de las herramientas que consideramos más relevantes:

agentTool III

Dentro de la categoría antes mencionada podemos destacar a agentTool III [36]. Es un entorno de desarrollo que asiste al usuario en analizar, diseñar e implementar sistemas multiagentes basándose en la metodología O-MaSE[37]. Entre los modelos a los cuales da soporte están "Goal model", "Agent Model", "Role Model", "Organizational Model", "Protocol Model", entre otros. Además la herramienta posee dos componentes adicionales: (i) un módulo de validación que verifica la consistencia de los modelos planteados como así también la validación entre modelos; (ii) un módulo específico que permite la generación de código para diferentes plataformas.

GAIA4E

GAIA4E[38] es el nombre que recibe la herramienta creada para la metodología GAIA[39]. Cubre todas las fases del proceso permitiendo al diseñador documentar los modelos correspondientes de acuerdo a una notación específica. Implementado como plugin para Eclipse Environment, define su perspectiva conformada por tres vistas principales: Eclipse Navigator, GAIA Properties y Image Viewer. Además, la herramienta ofrece la posibilidad de activar cualquiera de las tres fases definidas en el proceso permitiendo que el usuario elija dónde quiere trabajar.

IDK

Ingenias[40] también cuenta con una herramienta para el modelado de sistemas multiagentes llamado Ingenias Development Kit (IDK)[41]. Es un entorno visual rústico que permite la gestión del proyecto con múltiples diagramas, todos ellos organizados en paquetes, de acuerdo con la metodología. También, provee la posibilidad de crear repositorios de elementos, lo que facilita su reutilización.

PDT

Si bien Prometheus[42] es una metodología considerada del tipo Agent-Centred MultiAgent System (ACMAS), la herramienta construida para la misma es interesante de mencionar. Esta se denomina Prometheus Design Tool (PDT)[43] y provee al diseñador

del sistema de una interfaz gráfica intuitiva que facilita el desarrollo de varios de los artefactos definidos para la metodología. Adicionalmente, permite verificar la consistencia de los modelos propuestos realizando validaciones cruzadas entre varios modelos, propagación automática de los elementos de diseño cuando es posible y apropiado, etc.

PTK

La primera herramienta creada para la metodología PASSI[33] fue Metameth[44], cuyo proyecto fue abandonado por problemas de implementación. PASSI Toolkit(PTK)[45] es el nombre del nuevo proyecto, cubre todas las etapas de desarrollo de la metodología permitiendo al usuario realizar validaciones entre modelos. Los autores de la metodología como así también de PTK, actualmente están abocados en el refinamiento de ASPECS.

El objetivo de estas herramientas es la de proveer soporte visual y contribuir en la gestión de proyectos de las metodologías para las que fueron diseñadas. Sin embargo, la relación entre los diagramas que conforman el modelo y los diagramas implementados en las distintas CASE son variadas. Por ejemplo: tanto en agentTool III como PTK es posible la utilización de todos los diagramas definidos por sus respectivas metodologías, permitiendo así una documentación completa del sistema modelado. En cambio otras herramientas, tales como IDK o Rebel, y por diversos motivos, sólo cubren determinados diagramas (la mayoría cubre los diagramas principales).

En la actualidad la verificación de modelos es una característica obligatoria para cualquier herramienta dado que permite probar, automáticamente, que los modelos sean conceptualmente válidos tanto en su sintaxis como en su semántica. Las inconsistencias pueden ser detectadas de dos maneras: la primera, a través de la validación del modelo donde es posible encontrar inconsistencias en el mismo diagrama (perspectiva específica). Esta característica es provista por todas las herramientas consideradas para el análisis con excepciones de IDK y Rebel. La segunda característica es el Crosschecking que nos permite verificar si el diseño es internamente consistente entre los diagramas. Eso significa que los conceptos incluidos deben ser verificados para comprobar si son consistentes con la manera en el cual los otros diagramas los usan. Para las herramientas OCMAS descriptas, IDK, OpenTool y Rebel no poseen tan importante característica.

Un caso para tener en cuenta es el de GAIA4E; el cual cubre las etapas de análisis y diseño pero no la de implementación dado que la metodología GAIA no especifica la mencionada fase.

2.5. El Metamodelo Organizacional CRIO

CRIO[11](Figura 2.2) es un metamodelo basado en los conceptos organizacionales. El mismo deriva de la integración y extensión de dos metamodelos existentes. El primero es el metamodelo Role-Interaction-Organization(RIO)[46] que fue pensado para el modelado organizacional de sistemas multiagentes. El segundo es un framework para el modelado holónico de sistemas[47]. El metamodelo CRIO redefine muchos de los conceptos previamente definidos e introduce el concepto de capacidad[48].

CRIO está basado en el enfoque de Desarrollo Dirigido por Modelos(MDD). Este tipo de método es ampliamente utilizado en la industria del software. Esta situación se evidencia con la adopción en 2003 de los estándares de Arquitectura Dirigida por Modelos por el Object Management Group(OMG) y el creciente número de herramientas basadas en los principios detrás de este enfoque. El enfoque MDD pone al modelo en el corazón del proceso de diseño de software. Algunos de sus principios son: (i) la posibilidad de especificar el sistema objetivo independiente de la plataforma de implementación. (ii) especificar la plataforma de implementación y determinar una plataforma específica para el sistema; y finalmente (iii) transformar la especificación del sistema en una especificación compatible con la plataforma seleccionada.

Su nombre resulta del acrónimo formado por sus cuatro conceptos principales:

Una *Capacidad* es la descripción de un know-how/servicio. En otras palabras es la especificación de una transformación de una parte de un sistema o de su ambiente. Es una abstracción de alto nivel que promueve la reusabilidad y modularidad y en este sentido puede ser considerado como un componente básico de diseño. Además, el concepto de capacidad permite la definición de un rol sin hacer ninguna suposición de la arquitectura interna de este.

Un *Rol* es definido como un comportamiento esperado (un conjunto de tareas del rol ordenados por un plan) y un conjunto de derechos y obligaciones dentro del contexto de la organización. El objetivo de cada rol es contribuir en alcanzar los requerimientos de la organización dentro del cual está definido.

Una *Interacción* es una secuencia de eventos intercambiadas entre roles (una especificación de alguna ocurrencia que puede potencialmente disparar efectos en el sistema) o entre roles y entidades fuera del sistema.

Una *Organización* se define como una colección de roles y sus interacciones dentro de un determinado contexto.

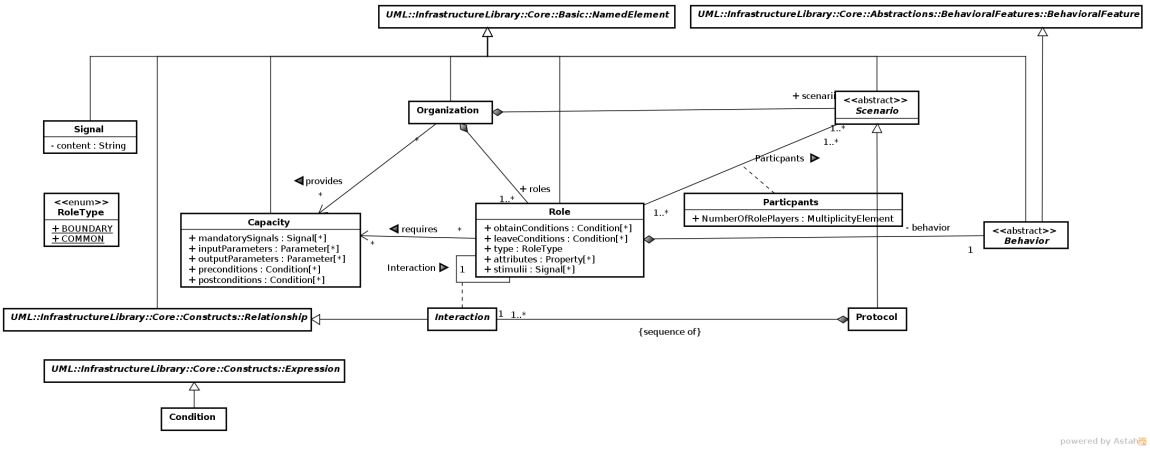

FIGURA 2.2: Metamodelo CRIO.

Por último, este metamodelo posee una base formal denominada OZS. Este lenguaje formal propuesto por [49] está compuesto de dos formalismos. Uno de ellos es Object-Z[50] el cual extiende Z, y el otro son los "Statecharts" de Harel[51]. El primero sirve para describir las estructuras de datos y funciones mientras que el segundo captura los aspectos de comportamientos y reactivos. La combinación de estos elementos es ideal para modelar un sistemas multiagentes, dado que permite la especificación formal del modelo y la verificación de las propiedades definidas para los conceptos[52].

2.6. La Metodología ASPECS

CRIO es la base fundamental de la metodología ASPECS[1], la cual define un proceso de desarrollo de software. En otras palabras, se establecen los pasos a seguir cubriendo áreas claves que van desde los requerimientos hasta la codificación permitiendo un modelado de sistemas con diferentes niveles de abstracción. ASPECS está inspirado en el enfoque Arquitectura Dirigida por Modelos (o mejor conocida por sus siglas en inglés como MDA; Model Driven Architecture) que define tres niveles de modelos donde cada uno de estos hace referencia a un metamodelo diferente. En la figura 2.3 se muestra la correspondencia entre los dominios de ASPECS y MDA.

En el nivel superior, y equivalente al *Modelo Independiente de la Computación*(CIM) de MDA, está presente el "Dominio del Problema" que provee una descripción organizacional del problema independiente de una solución específica.

El análogo al *Modelo Independiente de la Plataforma*(PIM) es el "Dominio de Agencia" que incluye los conceptos relacionados con una solución orientada a agentes para el problema analizado en la etapa anterior.

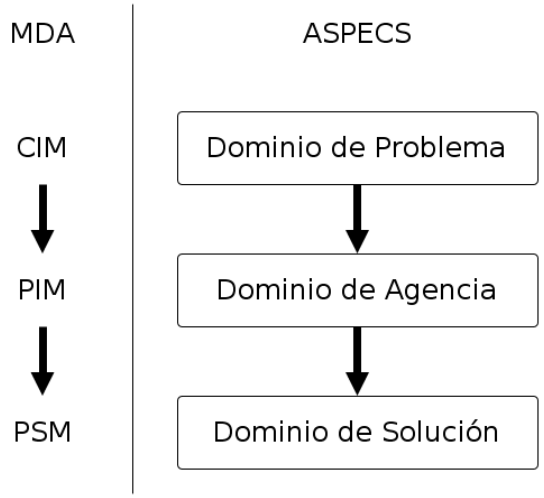

FIGURA 2.3: Dominios de ASPECS.

Por último, en el nivel inferior de ASPECS se encuentra el "Dominio de la Solución" que está relacionado con la implementación de la solución sobre una plataforma específica por lo que este dominio es dependiente de una plataforma de implementación particular. El mencionado dominio se corresponde con el *Modelo Específico de la Plataforma*(PSM) de MDA.

Marco general

La Figura 2.4 muestra el proceso de desarrollo de ASPECS que consta de tres fases: (i) Requerimientos del Sistema, (ii) Dominio de Agencia y (iii) Implementación e Implantación.

Requerimientos del Sistema

El modelo organizacional puede estar compuesto por organizaciones cuyo comportamiento global cumple con la especificación del sistema. La primera actividad a realizar es la *Descripción de Requerimientos del Sistema*, que permitirá recolectar e identificar los requerimientos utilizando técnicas clásicas como los Casos de Uso. El vocabulario o los términos utilizados por los usuarios deben ser identificados y plasmados en la *Descripción de la Ontología del Problema*. Por último, la actividad *Identificación de las Organizaciones* busca asociar los requerimientos a organizaciones. Cada una de las organizaciones definidas será responsable de mostrar un comportamiento que cumpla con algunos de los requerimientos. El resultado de esta última actividad es una primera aproximación que luego será, en futuros ciclos de desarrollo, expandida y refinada para obtener al final una organización jerárquica que represente la estructura y el comportamiento del sistema.

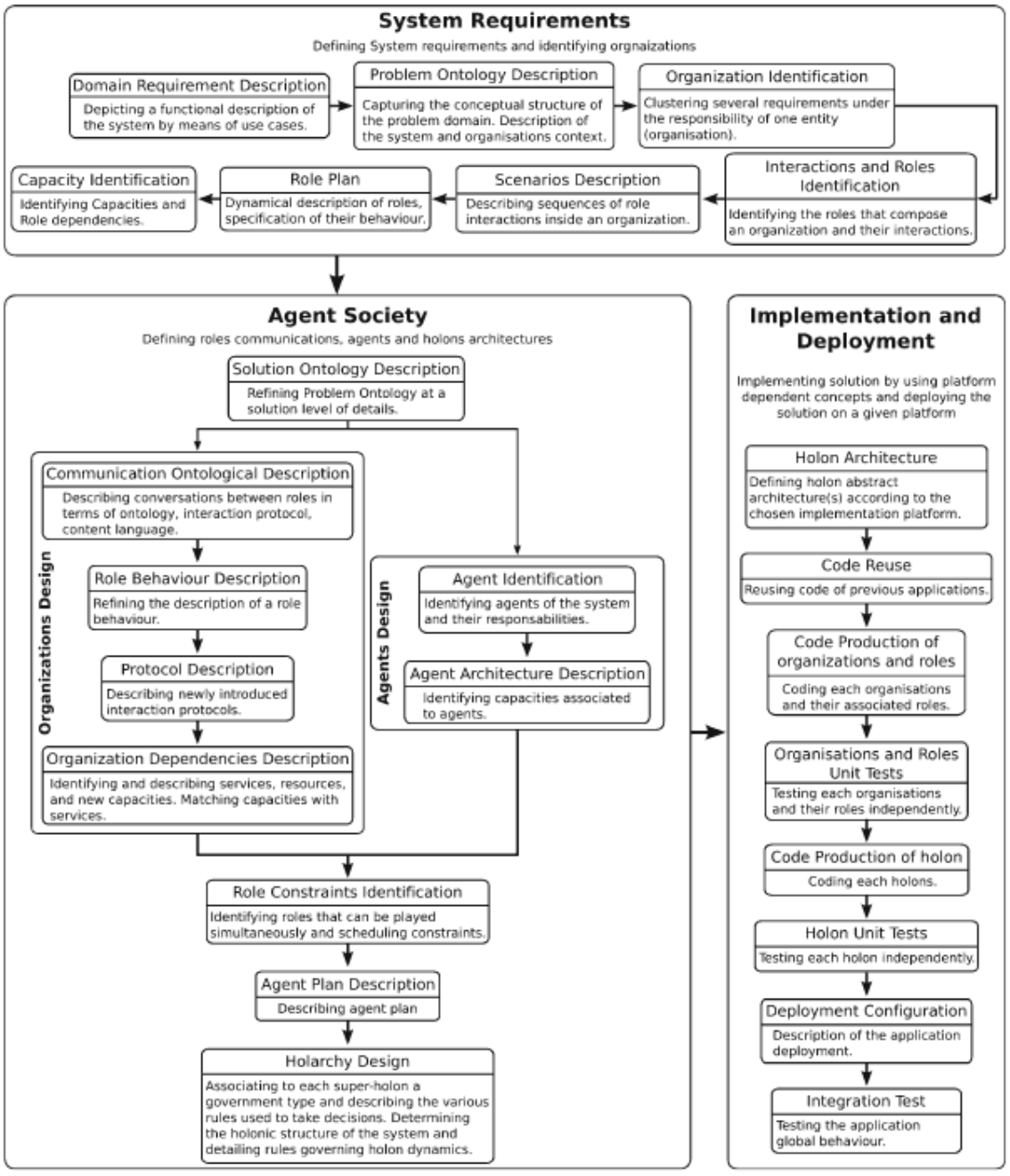

FIGURA 2.4: Proceso de ASPECS. (figura extraida de [1])

La *Identificación de Roles e Interacciones* procura descomponer el comportamiento global de una organización en comportamientos más pequeños. Cada uno de estos comportamientos identificados será exhibido por un rol. Además, las interacciones entre roles deben ser definidas, lo que permitirá a la misma organización proveer el contexto común.

La *Descripción de Escenarios* detalla la secuencia de interacciones entre los roles involucrados en cada uno de los escenarios. Esto se deduce de las descripciones textuales de un caso de uso del sistema, de los diagramas de ontología y de la actividad de identificación

de roles e interacciones. Se describen las interacciones identificadas entre todos los roles definidos para una organización dada.

El objetivo de la actividad *Plan del Rol* es proveer para cada rol una descripción del comportamiento necesario para poder satisfacer la parte de los requerimientos que le fueron delegados dentro de la organización.

El objetivo de la actividad *Identificación de la Capacidad* es definir el comportamiento genérico del rol identificando cuales son las competencias necesarias que debe tener el agente para poder jugarlo. En este sentido, decimos que una capacidad es la descripción de lo que una organización es capaz de hacer sin decir como lo hará.

Dominio de Agencia

Una vez concluida las actividades anteriores, se debe continuar con la fase de *Diseño de Sociedad de Agentes*. El objetivo principal es definir una solución orientada a agentes del problema, identificando los agentes, las interacciones y las dependencias existentes entre estos dos. El Holón es la parte central de la metodología ASPECS, que es una estructura auto-semejante compuesto a su vez de otros holones por lo que la solución estará principalmente compuesta de múltiples perspectivas. En una estructura jerárquica puede ser observada de acuerdo al nivel de abstracción ya sea o como una entidad autónoma o un grupo de holones interactuando. Al final de esta fase, la estructura organizacional es mapeada en una Holarquía donde cada una de las organizaciones identificadas en la fase anterior será instanciada en forma de grupo. Además, la última actividad consiste en definir un conjunto de reglas útiles para el proceso de toma de decisión ejecutadas dentro del cuerpo del holón.

Implementación e Implantación

Todo lo referente a la implementación de la solución orientada a agentes diseñada en la fase anterior es definida en la fase de *Implementación e Implantación*. Estos elementos en general pueden ser aplicados a diversas plataformas con poco o ningún cambio en la codificación; sin embargo, la plataforma que mejor se adapta es Janus[53] dado que fue desarrollado basado en gran medida a los metamodelos de ASPECS.

2.7. Conclusiones

En el presente capítulo se introdujo la noción de agente y de sistemas multiagentes. A su vez, dentro de los SMA, se destaca el interés creciente del enfoque organizacional como una nueva manera de modelar sistemas complejos. Nuestro interés se debe a dos motivos; por un lado, es posible proveer una solución evitando los modelos centrados en la arquitectura interna de los agentes tal como sucede en los enfoques tradicionales. Y por el otro, capitalizar los beneficios que puede aportar a la ingeniería de software orientada a agentes tales como la abstracción, modularidad, seguridad, reusabilidad, independencia de los lenguajes etc.

Se ha realizado una discusión de los distintos metamodelos basados en la metáfora social. También hemos repasado las distintas herramientas que han sido propuestas para el modelado de sistemas basado en el enfoque organizacional. Si bien la lista presentada no es exhaustiva, consideramos que las herramientas detalladas a lo largo de la sección son las más significativas dentro del ámbito académico y/o comercial.

Además, se detalla el metamodelo CRIO que fue adoptado para el desarrollo de esta tesis. Por último se presenta el proceso de desarrollo ASPECS, considerado uno de los mas completos en la actualidad[35], que define la secuencia de pasos sistemática y los actores involucrados en el desarrollo de un sistema multiagentes.

Capítulo 3

Entorno de Desarrollo

3.1. Infraestructura de Desarrollo

Para que un proyecto sea ejecutado exitosamente en una organización no tan solo se debe contar con personal capacitado sino que además se debe disponer de una infraestructura que permita gestionar adecuadamente el proyecto. Dicha infraestructura de desarrollo tiene una marcada importancia estratégica dentro cualquier organización dado que puede limitar o potenciar el crecimiento de los proyectos. Es por esto se debe formar recursos humanos que tengan un conocimiento del potencial y el impacto de la utilización de las nuevas tecnologías en los proyectos.

Los ámbitos de desarrollo de software agrupan y organizan un conjunto de tecnologías que sustentan los proyectos de diversas formas. Un diseño pobre o poco robusto de dicho ambiente no tan solo no aprovecha al máximo los recursos disponibles, sino que podría llevar a la deriva un proyecto de desarrollo al punto de producir un sistema de difícil mantenimiento, fuera del presupuesto, fuera de plazos de entrega, que no cumple con los objetivos de calidad acordados o incluso el fracaso total. La elección de dichos componentes no es una tarea trivial sino que debe ser una actividad meticulosa y detallada dado que debe existir entre estos, además de la prestaciones individuales, una integración armoniosa y fluida para evitar cualquier tipo de inconvenientes.

A lo largo del presente capítulo se presentarán las distintas herramientas seleccionadas como parte de la infraestructura de desarrollo para la aplicación que será introducida en el capítulo 4. Entre ellas están el sistema de versionado GIT, gestor de proyecto Maven, integración continua Jenkins y los frameworks de Eclipse como RCP, EMF, GMF.

3.1.1. Sistema de Control de Versión

El control de versiones es un proceso por el cual se pretende llevar un histórico o registro de todos los cambios que se realizan sobre un determinado archivo o conjunto de archivos. Esta técnica resulta imprescindible cuando se está desarrollando sistemas complejos o de gran envergadura dado que pueden presentarse situaciones en donde determinados cambios impactan de forma negativa en diferentes partes del sistema llevándolo a un estado no deseado. A su vez, la dinámica de desarrollo por parte de un grupo de personas que trabaja sobre un mismo módulo del sistema puede ser compleja y difícil de seguir incluso aún tratándose de un pequeño grupo de desarrolladores. Por este motivo, el versionado ayuda a tener un control sobre los cambios que puede llegar a realizar cada uno de los integrantes del equipo, de tal manera que un administrador de proyectos o la persona a cargo de la integración de módulos es la responsable de aceptar o rechazar determinados cambios.

Existe dos grandes tipos de versionado: Centralizado y Distribuido. Los sistemas centralizados, tales como Subversion, CVS o Perforce fueron los primeros en desarrollarse y rápidamente se convirtieron en estándares de la industria. Estos están compuestos por un servidor central que almacena todos los archivos que están siendo versionados. Entre las ventajas que podemos destacar de los mecanismos centralizados están: (i) La facilidad en la administración de los cambios dado que posee un control fino sobre lo que los desarrolladores pueden o no realizar. Esto se debe a que cuenta con un control de la base de datos local donde están registrados todos los usuarios. (ii) Todos los usuarios conocen en un cierto grado que es lo que están realizando sus compañeros en el proyecto promoviendo así una visión más global de sistema.

Sin embargo, este sistema no está exento de defectos. El problema más grave, denominado *Punto Único de Falla*, está dado por su principal característica: la centralización. Por ejemplo, si el servidor deja de funcionar, el tiempo que el mismo esté fuera de servicio ningún usuario podrá subir sus modificaciones o resguardar los cambios de los archivos sobre los cuales estaban trabajando. Peor aún, si la caída del servicio de versionado provoca el daño o la corrupción de archivos, y además no se han realizados los backups necesarios, existe la posibilidad de perder absolutamente todo.

En cambio en los sistemas distribuidos no existe un repositorio central sino que cada usuario cuenta con su propio repositorio local además del remoto, permitiendo intercambiar y mezclar los distintos archivos entre los usuarios. A su vez no necesitan estar conectados a una red para poder realizar las operaciones de salvaguarda de cambios. Ofreciendo con estas características mayor autonomía y rapidez (si el servidor remoto

queda fuera de servicio, las personas pueden seguir trabajando). En cuanto a la corrupción de los datos que pueda producirse en el repositorio remoto, o en el repositorio local, existe la posibilidad de recuperar la última versión del sistema de cualquiera de los usuarios dado que la información se encuentra replicada entre todos ellos. Sin embargo, lo mencionado no elimina por completo la necesidad de una política de backup.

En lo referente a tipos de sistemas de versionado distribuidos, podemos mencionar los dos más importantes: GIT y Mercurial.

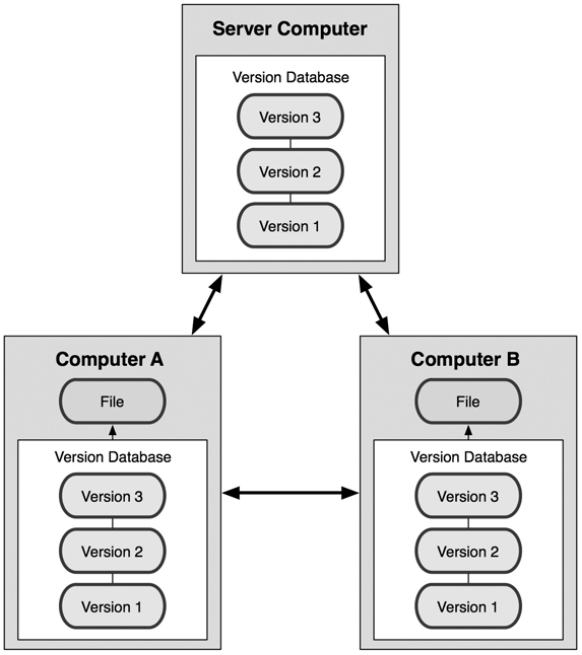

FIGURA 3.1: Arquitectura GIT.

GIT es un sistema de control de versiones desarrollado principalmente por Linus Torvalds y por la comunidad de desarrolladores del sistema operativo Linux. Para su creación se tuvo en cuenta características tales como velocidad, diseño simple, soporte robusto al desarrollo no-lineal, totalmente distribuido y habilidad de poder manejar proyectos de largo plazo, tal como el desarrollo del Kernel de Linux, de forma eficiente. En resumen, GIT emplea una combinación entre un sistema local, que hace referencia a que cada usuario tienen en su PC (Personal Computer) un servidor local para el control de versiones, y uno distribuido, dado que trabaja reflejando los cambios en un repositorio remoto tales como un servidor remoto o un servicio web como GITHUB[1] o Bitbucket[2]. A su vez, entre los usuarios particulares, y sin necesidad de pasar por un repositorio remoto, es posible compartir las modificaciones usando una comunicación peer-to-peer. En la imagen 3.1 se puede apreciar tal configuración de trabajo.

[1] http://www.github.com
[2] http://www.bitbucket.org

El motor de GIT, a diferencia de otros sistemas de versionado, trabaja realizando snapshots de la versiones que van cambiando en lugar de hacer únicamente una diferencia con respecto a la versión inmediata anterior. La ventaja de realizar el versionado de esta manera es que cada *snapshot* guarda el estado de los cambios de el/los archivos modificados. De no haberse realizado un cambio en un archivo al momento de realizar la tarea de versionado, GIT genera un enlace que hace referencia al archivo del *snapshot* anterior, tal como se muestra en la figura siguiente (Figura 3.2).

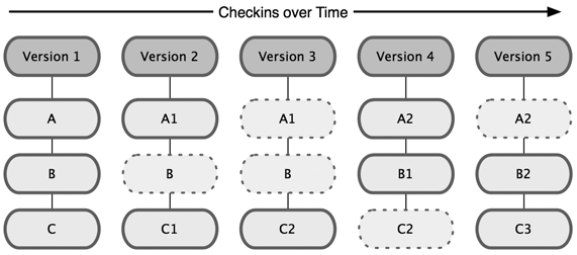

FIGURA 3.2: Manejo de versiones en Git.

En resumen, GIT es un sistema de versionado que trabaja en forma local, offline, standalone y a su vez distribuido que facilita enormemente la tarea de administrar el código desarrollado por un equipo de trabajo, manteniendo la integridad y coherencia de la información.

3.1.2. Herramienta de compilación automática y gestión de dependencias

Una problemática que se presentaba frecuentemente en los comienzos de la industria del software era la gestión de las bibliotecas de código de las cuales dependía una aplicación en desarrollo. A menudo si un desarrollador agregaba una librería dado que necesita algunas de sus funcionalidades y dicha modificación no era comunicada al resto del equipo, estos últimos potencialmente experimentaban problemas de compilación. Incluso si contaban con la información sobre la nueva biblioteca, debían contar también con la versión y las dependencias necesarias de la misma. No pasó demasiado tiempo hasta que en la industria aparecieron una serie de herramientas que facilitan la gestión de dependencias, entre las que podemos mencionar a Composer, Make, Ant, Maven, Gradle, etc.

Para nuestro proyecto optamos por Maven debido que se adapta más a nuestras necesidades. Sin embargo, no es posible realizar una definición sencilla de Maven dado que es más que solo una herramienta de gestión dependencias. Creado por Jason van Zyl en 2002, es una iniciativa de la Apache Foundation que permite administrar proyectos en distintos aspectos del software como es la gestión dependencias ya sea de bibliotecas,

con otros módulos, componentes externos, o administrar el ciclo de vida del proyecto, el orden de construcción e integración de elementos, generación de documentos, ejecución de test (testing) y hasta la posibilidad de realizar una implantación automatizada. Su elemento principal es el Project Object Management (POM), un archivo de configuración basado en XML, que actúa como un descriptor de proyectos en donde se registran sus atributos, instrucciones de como compilarlo y toda la información referente a las librerías que serán necesarias.

En Maven, los proyectos tienen asociado un ciclo de vida predefinido(Figura 3.3). Este ciclo de vida están constituido por varias fases.

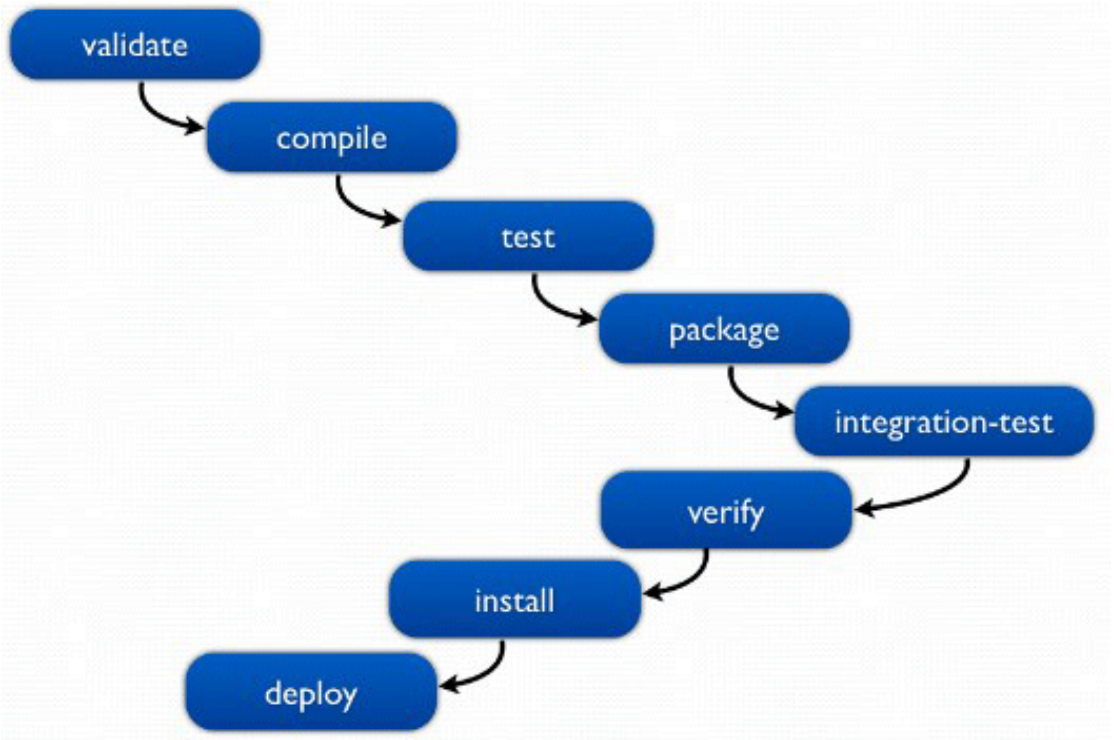

FIGURA 3.3: Ciclo de vida de un proyecto Maven.

- *validate*: Valida que el proyecto sea correcto y toda la información necesaria esté disponible.
- *initialize*[3]: Permite configurar propiedades del proyecto y la creación de la estructura de directorios.
- *compile*: Genera los ficheros .class compilando los fuentes .java.
- *test*: Ejecuta automáticamente los test JUnit existentes, abortando el proceso si alguno de ellos falla.

[3]No presente en la figura.

- *package*: Empaqueta el código compilado generando así el archivo distribuible .jar.
- *verify*: Verifica que el paquete generado es válido y cumple con los criterios de calidad.
- *install*: Copia el fichero .jar a un directorio de Maven de nuestro ordenador(repositorio local). Así, estos .jar pueden utilizarse en otros proyectos Maven en el mismo ordenador
- *deploy*: Sube el paquete a un repositorio Maven remoto para ser compartido con otros desarrolladores y/o proyectos.

A su vez, cada una de las fases de la lista anterior tienen asociadas un número variable de metas (*goals*) que permiten realizar una configuración más precisa de la ejecución de cada fase.

Otra característica de Maven es la posibilidad de crear una aplicación a partir de la utilización de una plantilla (la misma puede tener como finalidad una aplicación web, RCP, etc) que provee los paquetes básicos necesarios como así también la estructura estándar de directorios para el tipo de aplicación a desarrollar.

El funcionamiento de Maven es sencillo, el núcleo principal del sistema está diseñado especialmente para trabajar en red (Internet o Intranet). Una vez que se ha realizado la configuración apropiada del archivo POM, y se ha dado comienzo al ciclo de vida del proyecto, el sistema empezará a ejecutar cada una de las metas definidas en las fases (generalmente son dejadas por defecto). En cuanto a las dependencias declaras, el motor realizará una descarga automática de los plugins desde un repositorio remoto central disponible[4] logrando que todas las personas involucradas en el proyecto descarguen las mismas librerías, versiones y dependencias. Además, el repositorio mencionado cuenta con una vasta variedad de versiones de distintos proyecto de software "open-source" que pueden ser reutilizados en caso de ser necesario.

Por último, la posibilidad de utilizar modelos de producción de software que fueron probados por diversos proyectos (basado en el principio de convención sobre configuración) y la idea de reutilización de la lógica de construcción en vez de la lógica del código, han hecho de Maven nuestra elección para el desarrollo de nuestra herramienta.

[4]Generalmente es http://mvnrepository.com, pero es posible definir uno propio.

3.1.3. Integración Contínua

Uno de los momentos más críticos del ciclo de vida de desarrollo de un sistema es la integración de componentes (muchas veces desarrollados por diferentes programadores) y puesta en marcha de la aplicación recién integrada.

Martin Fowler establece que "la Integración Contínua (IC de ahora en más) es una práctica de desarrollo de software donde los miembros de un equipo integran sus trabajos de manera frecuente, usualmente cada persona hace al menos una integración diaria, llegando a realizar múltiples integraciones por día. Cada integración es verificada por una herramienta automatizada (incluyendo el test) para detectar errores lo más rápido/-temprano posible. Muchos equipos de desarrollo encuentran que este enfoque tiende a reducir significativamente los problemas de integración y permite desarrollar un software altamente cohesivo más rápidamente".

Para llevar a cabo una buena implementación de las técnicas de IC se deben adoptar una serie de prácticas tanto individuales como colectivas. Estas son:

- **Subir frecuentemente código al repositorio.** Es una de las prácticas centrales de la IC que promueve que se deben realizar pequeños cambios e inmediatamente subir estas modificaciones al repositorio y no esperar hasta tener grandes modificaciones.
- **No subir al repositorio código que no compila.** Representa uno de los errores más frecuentes en los grupos de trabajo. Es por ello que se debe hacer hincapié en que los desarrolladores deben realizar sus compilaciones localmente antes de subir el código al repositorio.
- **Resolver los errores de compilaciones de manera privada antes de subir código al repositorio.** Una vez que se ha detectado e informado un error, el desarrollador debe inmediatamente resolverlo. Debido a las características de las prácticas, el error descubierto será mínimo y debe estar en las prioridades del proyecto.
- **Escribir pruebas de desarrollo automatizadas.**
- **Ejecutar compilaciones privadas.** Los desarrolladores deben compilar de manera local para evitar las compilaciones fallidas. Esta compilación permitirá obtener desde el repositorio los últimos cambios y compilar localmente con lo cambios recientes reduciendo el riego de poner en el repositorio código que no compila.

- **Evitar obtener código que no compila** Los miembros del equipo no deben obtener del servidor una compilación que no se ejecute hasta que el desarrollador responsable de este contratiempo resuelva el problema.

En la Figura 3.4 se puede observar la arquitectura que hasta en momento fue descripta en esta sección. Con el propósito de un mejor entendimiento, en la imagen se pueden observar dos servidores distintos (control de versión e IC) aunque en la práctica es posible que ambos actúen como servicios de un mismo servidor. El funcionamiento de los mismo se describirá a continuación

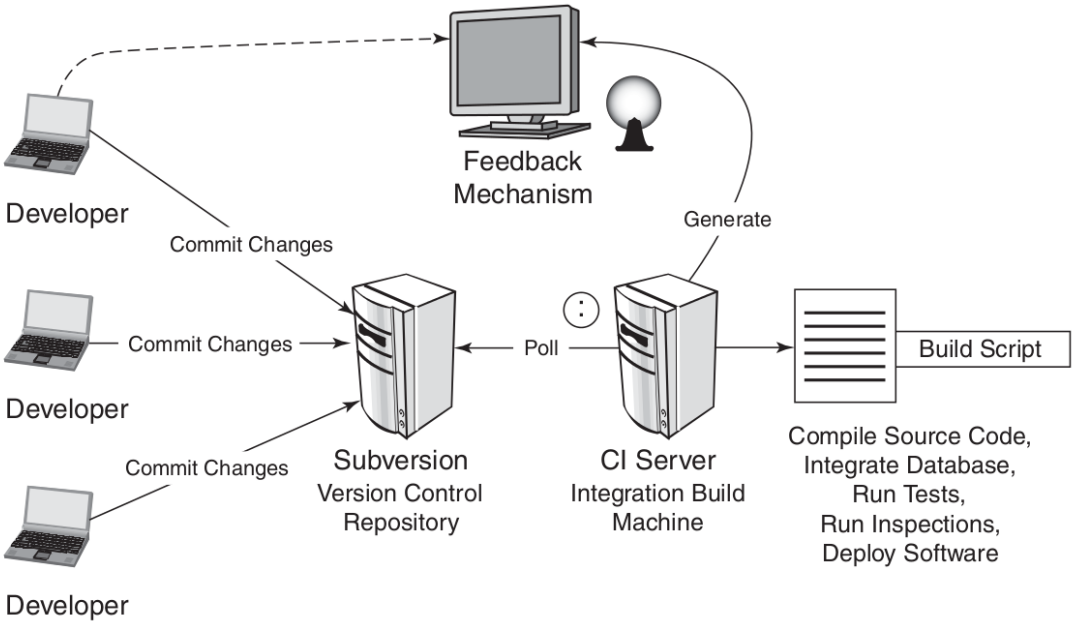

FIGURA 3.4: Manejo de versiones en GIT.

Si las tareas asignadas a los desarrolladores requieren un tiempo considerable para ser llevadas a cabo puede ser conveniente planificar las actividades, descomponiendo dichas tareas en sub-tareas de menor tamaño. Cada vez que se logra alcanzar cada una de estas metas, el desarrollador deberá realizar un "commit" para salvaguardar en su repositorio local las modificaciones realizadas. Este comportamiento se repetirá tantas veces como sub-tareas definidas existan y sólo entonces se estará habilitado para realizar un "push" de su repositorio local al remoto para compartir los cambios con su equipo. En este caso el repositorio remoto funciona como un nodo de sincronización dado que si un programador necesita de las modificaciones realizadas por otras personas sólo le bastará fijarse en el registro de modificaciones del repositorio. En caso de que existan tales modificaciones, el siguiente paso será realizar un "pull" para obtener una copia en nuestro repositorio local.

El funcionamiento IC es bastante diferente en la forma de interactuar con el usuario; generalmente es asociado a un demonio[5] que realiza un polling (sondeo) del repositorio remoto para verificar si se han realizado modificaciones. En caso de haberlas, el IC ejecuta una serie de actividades para comprobar que todos los elementos, previamente definidos, estén presentes y cumplan con las restricciones impuestas. Algunos ejemplos de estas actividades son: la documentación del código, integración a la base de datos, ejecución exitosa de los test e implementación del software, entre otras . En caso de que el "commit" supere exitosamente las restricciones pasará a formar parte del sistema que puede ser implementado. Por otro lado, si alguna de estas restricciones no se satisfizo, por ejemplo un test dio un resultado negativo, el servicio rechazará el "push" y notificará inmediatamente al propietario de los cambios para que realice las modificaciones pertinentes del caso.

Tal como menciona [54], la IC permite:

- **Reduce o mitiga riesgos.**
- **Reduce procesos manuales repetitivos.**
- **Genera un software ejecutable en cualquier momento y en cualquier lugar.**
- **Permite una mejor visibilidad del proyecto.**
- **Establece una mayor confianza en el producto de software del equipo de desarrollo.**

Es importante remarcar que esta técnica no necesariamente necesita de un herramienta de software automatizada sino más bien son actividades que las personas que forman parte del desarrollo deben incorporarlas en la cotidianidad de su trabajo.

3.2. Plataformas

3.2.1. Rich Client Platform

En los últimos años, la popularidad de las plataformas para desarrollar aplicaciones de escritorio ricas o enriquecidas se ha incrementado gracias a la experiencia y velocidad de desarrollo que ofrecen a los ingenieros de un determinado dominio. Este término, "rich client" o cliente rico, fue acuñado en los 90' y propone una marco de desarrollado

[5] En la jerga informática es un tipo especial de proceso informático no interactivo, es decir, que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario.

que promueve la reutilización de elementos y una alta personalización de la aplicación, además de la posibilidad de construir tales aplicaciones rápida y fácilmente. En otras palabras, se desarrolla en base a la reutilización de componentes dado que la mayoría de las aplicaciones tienen características similares como pueden ser los menúes, barras de herramientas, barra de estados, navegadores, etc, como así también las metáforas nativas tales como las acciones permitidas dentro de la aplicación (Drag and Drop, Clipboard, Workbeanch). Finalmente, esto permite que el desarrollador en vez de escribir una aplicación desde cero destine ese tiempo ahorrado en la definición de la lógica del dominio.

Eclipse Rich Client Platform (RCP)[55], desarrollado por Eclipse Foundation[6], provee un entorno rico en opciones y debido a su popularidad posee una comunidad importante que le da soporte.

Algunas de las características presentes en RCP son:

- **Componentes**: Todo está desarrollado basado en el concepto de plugins que permite una alta modularización del sistema. Un plugin es definido como una unidad de modularidad en Eclipse, de hecho todo en Eclipse es desarrollado bajo este concepto. Básicamente, un plugin es autodescriptivo y explicita una lista de los otros plugins a los cuales depende para un funcionamiento adecuado. Esta alta modularidad permite que las aplicaciones desarrolladas sobre RCP puedan ir evolucionando a través de un sencillo reemplazo de componentes, ya sea por la aparición de nuevas versiones u otro con mayores prestaciones. Además, RCP brinda una manera clara y sencilla de gestionar los plugins para construir aplicaciones complejas y funcionales.
- **Middleware e Infraestructura**: Cuenta con un colección de frameworks que facilitan el desarrollo de aplicaciones personalizadas, como por ejemplo: el uso de un paradigma flexible para el diseño de interfaces de usuario (UI, User Interface), UI's escalables, aplicaciones extensibles, soporte de ayuda, actualizaciones de red, manejo de errores y mucho más.
- **Portabilidad**: RCP brinda la posibilidad de ejecutarse sobre entornos pertenecientes a diferentes tipos de sistemas operativos, tales como los de PCs tradicionales hasta los móviles como los que poseen las tabletas, asistentes digitales (por ejemplo PDA, Personal Digital Assistant), teléfonos inteligentes. Todo esto es gracias a la Maquina Virtual de Java (JVM) sobre la cual se ejecuta.
- **Soporte para el desarrollo de herramientas**: Eclipse RCP brinda un conjunto de herramientas que permiten construir aplicaciones adaptas usando como base el

[6] https://eclipse.org/org/foundation/

mismo Eclipse RCP. Este conjunto de framewoks ayudan a los profesionales a diseñar, codificar, realizar testing y empaquetado de las aplicaciones específicas para un entorno.

- **Librería de Componentes**: RCP posee un conjunto de componentes con el cual el diseñador posee una base de elementos ya desarrolladas que puede agregar directamente a sus aplicaciones, facilitando y agilizando cada vez más el desarrollo. La comunidad de Eclipse constantemente desarrolla y deja a disposición del desarrollador; Plug-ins de UI, soporte para instalación y actualización, editores, consolas, frameworks de edición gráfica, frameworks para la manipulación de datos y muchos más.

Todas estas ventajas, las cuales resultan muy atractivas para un desarrollador de software, hacen de Eclipse RCP una de las opciones más utilizadas dentro del mercado en lo que se refiere a plataformas que se pueden llegar a utilizar para el desarrollo de aplicaciones personalizadas y de grandes prestaciones.

3.2.2. Modelado

Eclipse Modeling Framework[7] (EMF) es parte importante de una gran familia de proyectos llamado Eclipse Modeling Project[8](EMP). Es una librería que permite a los expertos construir y personalizar sus propias herramientas basadas en un metamodelo de datos estructurados denominado *Ecore*. Un *Ecore* simplifica la definición de un metamodelo facilitando la programación requerida para la implementación de un Modelado Específico del Dominio (o Domain Specific Language). Además, el uso de EMF permite el modelado de jerarquías donde un modelo es metamodelo de otro.

Los conceptos usados en este framework son mucho más simples (abstractos) que aquellos definidos por el Object Management Group[9] (OMG). Esta característica tiene como finalidad abarcar una vasta variedad de situaciones, además de obtener rápidamente un código ejecutable. Algunos de los elementos más importante del *Ecore* son:

- **EPackage** representa un paquete.
- **EClass** representa una clase, el cual puede contener un conjunto de atributos (EAttribute) y referencias (EReferences).
- **EAttribute** es un atributo que posee nombre y tipo.

[7]Eclipse Modeling Framework, www.eclipse.org/emf/
[8]Eclipse Modeling Project, www.eclipse.org/modeling/
[9]Object Manaagement Group, www.omg.org

- **EReference** es utilizado para asociar clases. Apunta a una clase referenciada y puede representar un contenedor.
- **EDataType** representa un tipo dato que puede ser tomado por un atributo.

Existen distintas formas de definir un modelo en EMF que incluye representaciones gráficas, anotaciones de JAVA, XMI, y archivos generados por algunas aplicaciones de UML. Sea cual sea el origen de la información para el armado del metamodelo, la representación que Eclipse EMF hace de la misma es en forma de árbol como se puede observar en la Figura 3.5. El árbol irá creciendo a medida que volquemos los conceptos que conformarán los términos definidos para el dominio. Si bien el formato de árbol puede resultar un poco engorroso cuando se está desarrollando una aplicación en donde intervienen un alto número de conceptos, el entorno cuenta con la posibilidad de representar este árbol y sus relaciones usando el formato de diagrama de clases tal como veremos en la sección 4.4.

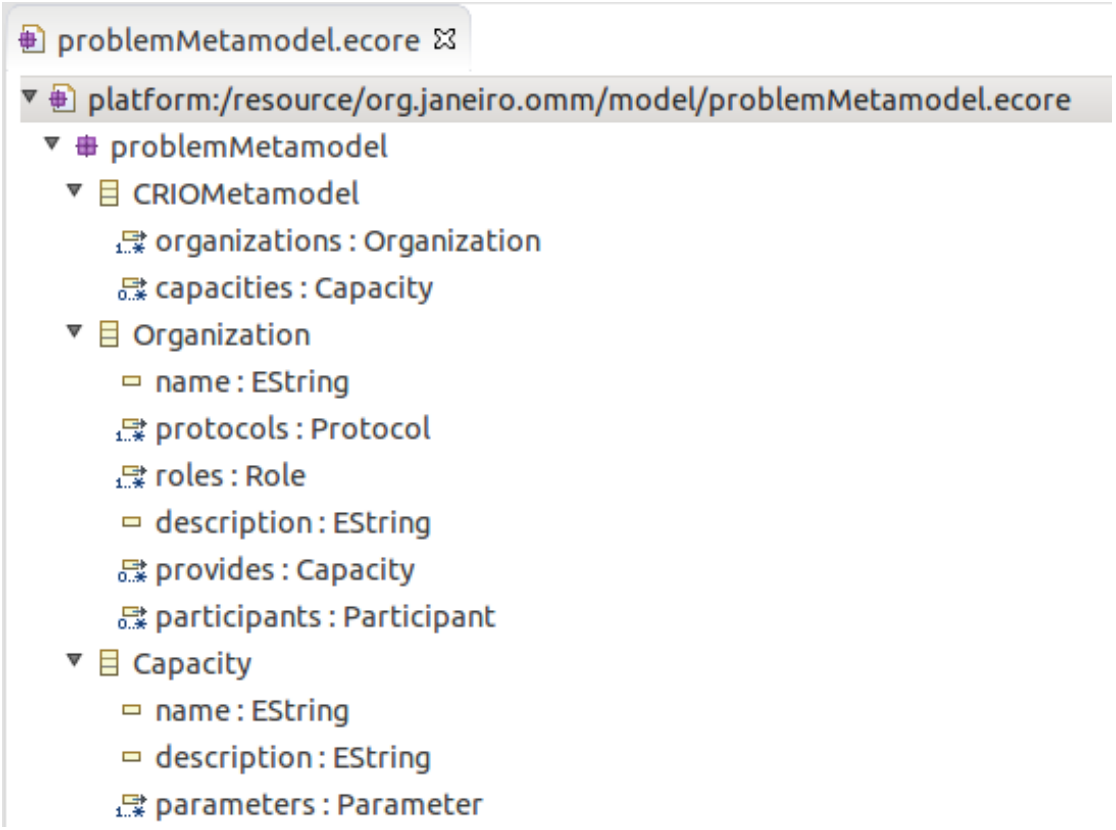

FIGURA 3.5: Ecore en forma de árbol.

Una vez concluida la etapa de definición del *Ecore*, el siguiente paso será definir las vistas (Editores) que tendrá el metamodelo. Para generar una de estas vistas, es necesario identificar el elemento raíz dentro del metamodelo y a partir del cual se podrá desarrollar un editor que permitirá manipular los elementos contenidos en el *Ecore*.

3.2.3. Parte Gráfica

Eclipse proporciona desde sus inicios un framework gráfico denominado Graphical Editor Framework (GEF de ahora en más)[10]. No obstante su uso para la construcción de editores gráficos es bastante complejo, puesto que es un entorno genérico e independiente del modelo subyacente además de poseer una extensa librería para personalizar los editores. El problema de la personalización de modelos radica en que gran parte del esfuerzo de implementación se invierte en mecanismos para tratar con ellos, así como en tareas de carga/serialización. El siguiente paso lógico en el desarrollo de los editores fue adoptar el uso de EMF como framework de modelado subyacente. No obstante, la integración de EMF y GEF no es trivial, y presenta diversas dificultades técnicas. Son por estos motivos que Graphical Modeling Framework(GMF) surge como un framework capaz de unificar ambos entornos, siguiendo una filosofía dirigida por modelos que simplifica significativamente los esfuerzos necesarios para el desarrollo de aplicaciones basadas en metamodelos.

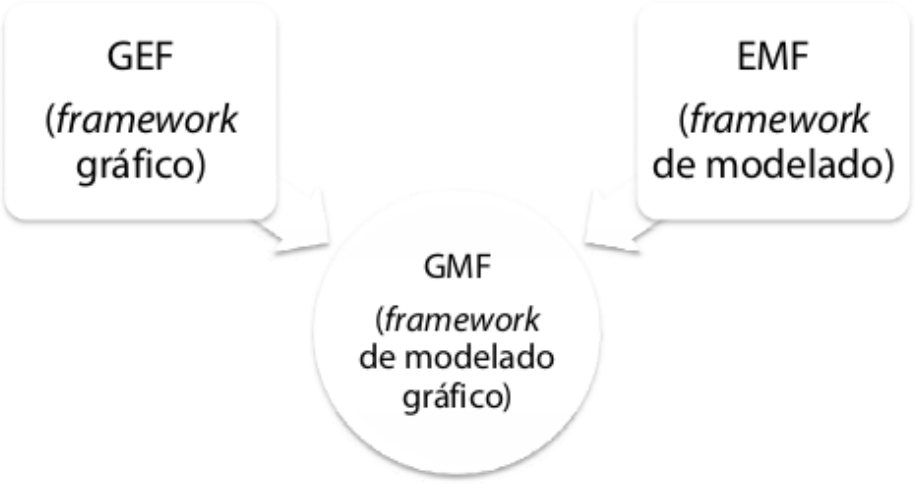

Figura 3.6: Graphical Modeling Framework.

El Dashboard de GMF (figura 3.7) define una secuencia de actividades necesarias para la creación de editores personalizados basado en el paradigma Model-View-Controller (MVC). Esto permite que una vez definido el DSL(Domain Specific Language, Lenguaje Específico del Dominio) para un dominio en particular, se pueda empezar a diseñar la metáfora gráfica que tendrán las primitivas del Ecore mediante el *Modelo de Definición Gráfico*(Graphical Definition Model). La definición gráfica consiste en decidir qué primitivas del modelado harán la función de nodos (elementos de la herramienta de modelado que se desea construir), cuáles serán conectores (enlaces entre los elementos de la herramienta de modelado) y cuáles etiquetas (propiedades de los elementos y de los enlaces).

[10]https://www.eclipse.org/gef/

Además, define qué representación gráfica (sus formas, dimensiones y colores) tendrán las primitivas como así también los posibles valores por defecto de sus atributos.

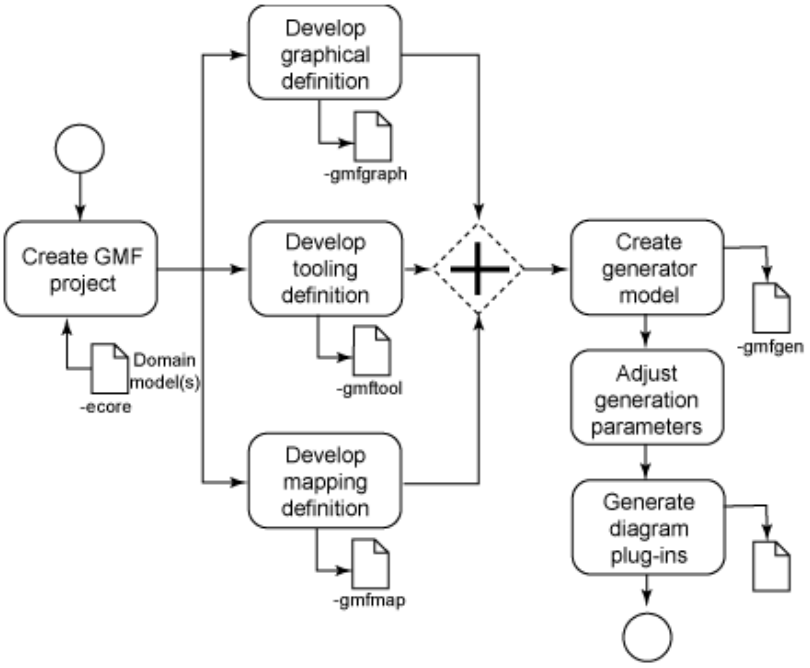

FIGURA 3.7: Graphical Modeling Framework.

La creación y especificación de los elementos del editor se ha de realizar después de la definición de la metáfora gráfica utilizando el *Modelo de Definición de Herramientas*(Tooling Definition Model). Estos elementos pueden ser menúes contextuales, definición de menúes que se agregarán al principal, la barra de botones y, el más crítico de todos, el panel de herramienta. Consideramos al panel de herramientas como la paleta que permite volcar, mediante una funcionalidad del tipo "drag and drop", las primitivas del modelo en los editores definidos. En este paso también se definen los iconos del la herramienta de modelado, que constituyen la iconografía de la paleta.

3.2.4. Integración

Completadas las actividades referentes a la metáfora gráfica y la especificación de los elementos del editor, se debe proceder con la definición de la correspondencia entre elementos del Ecore y gráficos. Es decir, se establecen relaciones ternarias entre los conceptos que conforman el Ecore, la metáfora gráfica y la definición de las herramientas. En esta actividad todo lo creado hasta el momento cobra sentido, dado que se relacionan todos y cada uno de los elementos definidos en las actividades anteriores. Este paso es el

último del proceso de creación antes de la creación del *Modelo de Generación de Código* del editor.

Cuando se ha completado el mapeo (y realizadas las modificaciones pertinentes), se puede proceder a la generación del Modelo de Generación o GMFGen. Este archivo de configuración permitirá ajustar los parámetros finales del plugin tales como la ubicación en los directorios relativos, nombre del plugin, nombre de los paquetes, ubicación física, etc.

Así, y habiendo particularizado el plugin según especificaciones del usuario, se proceda a la generación del código del editor gráfico en sí. Dos tipos de editor gráfico son posibles con GMF: un editor gráfico integrado como plugin de Eclipse o como una aplicación RCP (Rich Client Platform) del tipo stand-alone, que consiste en una aplicación autónoma. Queda a decisión de los diseñadores qué tipo de aplicación creerán conveniente generar.

3.3. Conclusiones

En este capítulo se ha introducido todo lo referente al conjunto de técnicas y aplicaciones que conforman el ambiente de desarrollo y que son necesarias para la correcta ejecución del proyecto. Para una mejor lectura se ha dividido el capítulo en dos secciones claramente diferenciadas.

En la primera (Sección 3.1) se realiza una descripción de la infraestructura de desarrollo. En ella mencionamos: (i) el sistema de control de versiones, que permite llevar un histórico de las modificaciones que se están realizando en la aplicación. Para nuestro desarrollo hemos elegido al sistema denominado GIT, dado que una de sus características más importantes es el versionado distribuido. (ii) El sistema de compilación y gestión de dependencias Maven el cual asegura una unificación de los criterios de compilación y empaquetado de software como así también garantiza que todos los colaboradores trabajen sobre las mismas versiones de las dependencias. (iii) las técnicas de integración continua establecen un conjunto de buenas prácticas a seguir para incrementar las posibilidades de éxito en el desarrollo de un sistema. Usamos Jenkins para correr los test bien se detectan modificaciones en el código principal y por el sistema de devolución de feedbacks acerca del estado del sistema. Estas y otras características hacen de la integración continua una herramienta imprescindible en la actualidad.

La sección 3.2 introduce los distintos elementos que conforman la plataforma de desarrollo para nuestra aplicación. Entre ellas están: (i) Eclipse RCP que permite la construcción de aplicaciones “ricas” en características haciendo uso intensivo de la reutilización de

recursos y componentes. Lo anterior se debe a que las aplicaciones en la actualidad poseen un patrón de similitudes evitando así construirlas desde cero. (ii) EMF, que permite definir metamodelos de datos estructurados basados en conceptos más abstractos que los definidos en la OMG y que representa el punto de partida para el desarrollo de una aplicación para un dominio en particular. (iii) GMF, que es un framework que permite definir las figuras (formas y colores) que representan los conceptos del metamodelos y que serán útiles para la definición de los editores donde serán manipuladas las instancias del metamodelos, en otras palabras nuestros modelos.

Consideramos que en la actualidad es imposible no contar con algún tipo de infraestructura de desarrollo mas allá del tamaño del proyecto y del número de integrantes. Estas infraestructuras permiten a los arquitectos de software gestionar ordenadamente tanto los recursos humanos como la marcha del proyecto llevando a las aplicaciones desarrolladas a un estándar alto de calidad.

Capítulo 4

Janeiro Studio

4.1. Introducción

Una Computer Aided Software Engineering (CASE de ahora en más) es un conjunto de herramientas informáticas que asisten al diseñador en algunas de las actividades relacionadas con el desarrollo de un sistema (requerimientos, análisis, diseño, codificación y pruebas). Estas herramientas permiten transmitir lo más rápido posible las intenciones de los desarrolladores mediante una combinación de notaciones gráficas y textuales que representa algún lenguaje específico compartido por el equipo de desarrollo. Además, incrementan la productividad de los equipos de desarrollo reduciendo tiempo y costos a su vez que mejora la calidad del software entregado[56].

En el presente capítulo presentaremos los avances realizados en la herramienta CASE Janeiro Studio[57]. Si bien el objetivo final de la mencionada herramienta será proveer de un soporte adecuado tanto para el metamodelo organizacional CRIO como la metodología ASPECS[1], actualmente Janeiro se encuentra en sus primeras etapas de desarrollo abarcando sólo la primera fase de la metodología. Desde su inicios fue concebido como una herramienta CASE multiplataforma de libre distribución, open-source, completamente desarrollada en JAVA que facilita el modelado de sistemas multiagentes basado en el enfoque organizacional.

En la Figura 4.1 se presenta el Entorno de Desarrollo Integrado (en inglés IDE, Integrated Development Environment) propuesto para Janeiro. En realidad, la terminología correcta en el mundo de Eclipse RCP es “perspectiva” y es un contenedor visual de un conjunto de “Vistas” y “Editores” presentadas al usuario. A saber:

(A) Explorador de Proyectos. Contiene a todos los proyectos que fueron creados en el espacio de trabajo. Además, cada proyecto es estructurado en forma de árbol

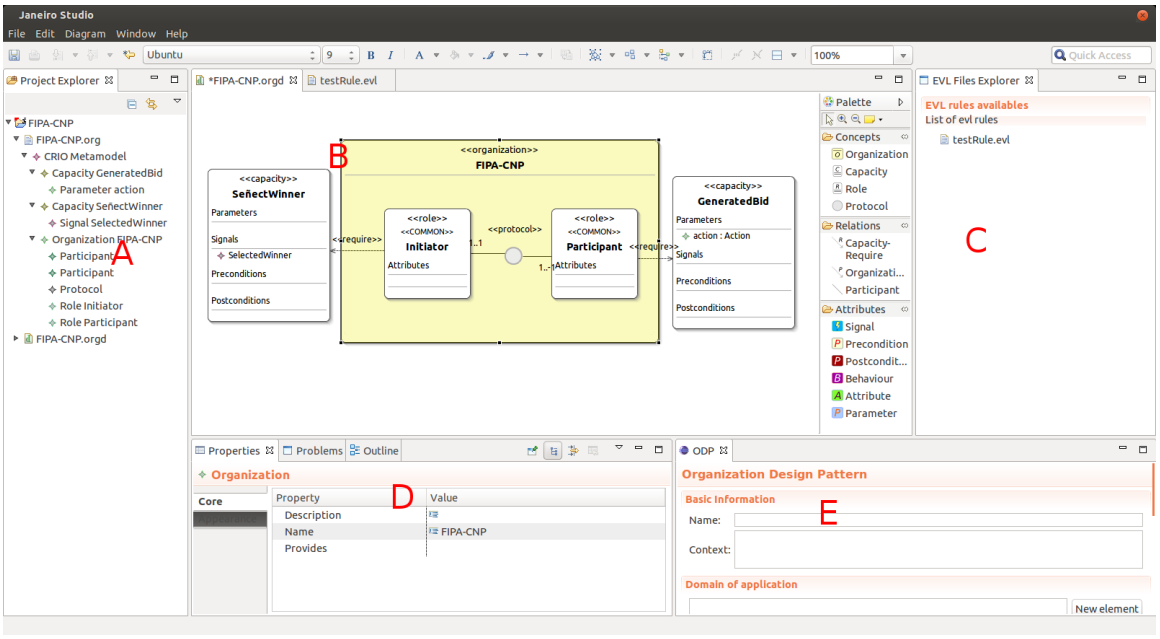

FIGURA 4.1: IDE del prototipo Janeiro Studio.

conformado por los elementos definidos por el usuario que están presentes en los diagramas. Esta estructura facilita la navegación y búsqueda de los conceptos plasmados en los distintos diagramas.

(B) Editores. Actualmente fueron definidos cinco editores para la manipulación de los Ecores. Los mismos serán explicadas más adelante en esta sección. Para cada uno de los tipos de diagramas se despliega una paleta con accesos directos a los conceptos gráficamente representados que pueden utilizarse para el armado del modelo.

(C) Explorador de reglas de validación. Janeiro cuenta con un conjunto de reglas de validación integradas, las mismas son para la validación sintácticas además de ser genéricas para todos los proyectos. Con esta vista también es posible definir reglas que atienden a situación particulares de cada proyecto.

(D) Conjuntos de *View-Part* (*Properties*, *Problems*, *Outline*). El primero de ellos, *Properties*, es posible especificar propiedades de los conceptos que pueden figuran o no en la notación gráfica. El segundo, *Problems*, muestra los errores arrojados por las validaciones sintácticas realizadas, y en un futuro las semánticas, aplicados en los modelos. Por último *Outline* nos permite tener una vista global del diagrama sobre el cual se está trabajando.

(E) ODP (*Organization Design Pattens* plugin, actualmente en desarrollo). La idea detrás de la creación de esta vista es brindar la posibilidad de registrar patrones de diseños que puedan presentarse en el modelo como así también la posibilidad de utilizar dichos patrones en nuevos diseños.

El núcleo principal de Janeiro es el Ecore creado con el framework EMF. En él se definen los metamodelos que son necesarios para representar un dominio en particular. Así los diagramas pueden considerarse "vistas" de distintas partes de un mismo modelo. EMF permite la creación dinámica de modelos definidos según un Ecore específico.

Para la manipulación de los Ecores, actualmente Janeiro Studio provee soporte para los cinco diagramas más relevantes del metamodelo CRIO, que además son los diagramas necesarios para cubrir la fase *Dominio del Problema* de ASPECS (Figura 2.3). Estos diagramas son: *Diagrama de Requerimientos del Dominio*, *Diagrama de Ontología*, *Diagrama Organizacional*, *Diagrama de Interacción* y el *Diagrama de Comportamiento*.

Debido a las diferencias entre los conceptos subyacentes del modelado, se tomó la decisión de definir tres Ecore independientes y vincularlos posteriormente. De esta forma, existen metamodelos específicos para el *Diagrama de Requerimientos del Dominio* y del *Diagrama de Ontologia del Problema*, mientras el metamodelo central a considerar es el del *Dominio del Problema*. Estas relaciones se encuentran resumidas en el cuadro 4.1.

Cuadro 4.1: Ecores y sus diagramas

Metamodelo	Diagramas	Descripción
Requerimientos	*Diagrama de Requerimientos*	Con diagramas similares a los usados en los Casos de Uso de UML, es posible documentar las expectativas y necesidades de los interesados. Estos diagramas permiten capturar los requerimientos funcionales y no funcionales, utilizando un lenguaje específico del dominio provisto por los usuarios.
Ontología	*Diagrama de Ontologías*	Permite identificar, modelado como un diagrama de clases, los conceptos, predicados y acciones relevantes del dominio. La idea principal es conceptualizar los requerimientos descriptos en el Diagrama de Requerimientos del Dominio o en cualquier documento que describa el sistema a desarrollarse.
Dominio del Problema	*Diagrama Organizacional*	Permite representar gráficamente la estructura organizacional. La idea detrás de este diagrama es modelar la organización que represente el comportamiento global. Esta representación es realizada a través de un conjunto de roles presentes en la organización y de la interacción entre ellos. Las capacidades asociadas a los roles definen el comportamiento que es requerido por el agente para que pueda tomar un rol. Los conceptos usados en este diagrama son : *Organización*, *Rol*, *Capacidad*, *Protocolo* entre otros.
	Diagrama de Interacción	Describe la secuencia de intercambio de información entre los roles que tienen lugar en un protocolo. Los conceptos más relevantes en este diagrama son: *Protocolo*, *Rol*, *Interacción*, *CallCapacity*, *Señales*, *Atributos*, etc.
	Diagrama de Comportamiento	Describe el comportamiento de un rol (la dinámica de la instancia). En otras palabras, permite representar los estados y transiciones posibles en un rol. Este diagrama es un perfil del diagrama de transición-estados de UML.

4.2. Metamodelos de Requerimientos y de Ontología

La mayoría de las metodologías para sistemas multiagentes existentes en la actualidad son procesos de desarrollo de software dirigidos por requerimientos. Esto significa que en las primeras etapas de las mismas están relacionadas con la captura de requerimientos tanto funcionales como no funcionales. El objetivo principal de la actividad *Descripción de Requerimientos del Dominio* es capturar la necesidades y expectativas de los interesados además de proveer un completa descripción del comportamiento de la aplicación bajo estudio. Esta actividad es generalmente llevada a cabo adoptando los *Diagramas de Casos de Usos* para la descripción de los requerimientos funcionales. En cambio, en los requerimientos no funcionales son utilizadas las anotaciones tradicionales de texto dentro de la misma documentación de los Casos de Uso.

En la Figura 4.2 se muestra el Ecore propuesto para el *Diagrama de Requerimientos del Dominio*. El mismo está conformado por *frames* que indican los límites del sistema o subsistemas. Afuera de estos límites se pueden definir a los usuarios externos que interactúan con el sistema. Dentro de los mismo, se definen los Casos de Uso que son comportamientos o descripciones de lo que debe hacer el sistema a desarrollarse.

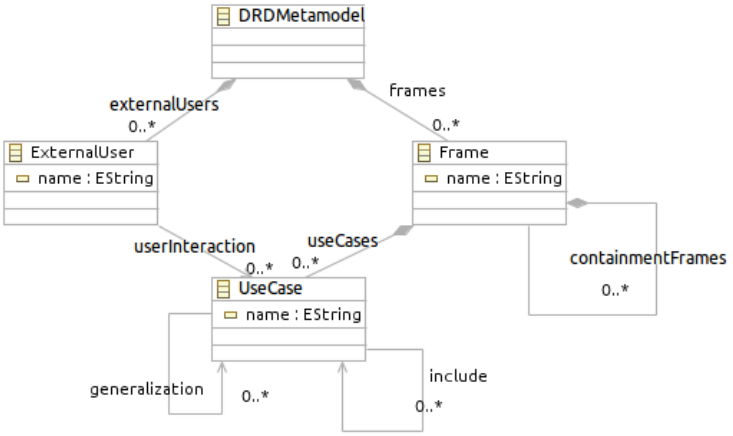

FIGURA 4.2: Metamodelo para el Diagrama de Requerimientos del Dominio.

La actividad *Descripción de la Ontología del Problema* permite capturar la vista general del dominio del problema. El motivo por el cual fue incorporada dicha actividad es que a que las personas involucradas en el desarrollo expresan sus requerimientos utilizando el lenguaje natural basados en sus propios términos y con un conocimiento implícito de su propio trabajo. El éxito de esta actividad dependerá del entendimiento que se tenga del problema a partir de lo realizado en la *Descripción de Requerimientos del Dominio*[15].

Para cubrir la actividad detallada en el párrafo anterior fue necesario implementar en Janeiro Studio un segundo metamodelo denominado PODMetamodel (Figura 4.3). En el

mismo se puede ver que el *Diagrama de Ontología* gira en torno a la clase *Concept*, el cual puede representar tanto un concepto, un predicado o una acción. En la figura también se pueden apreciar los distintos tipos de relaciones que es posible hacer: *composición*, *agregación*, *asociación* y por último, *generalización*.

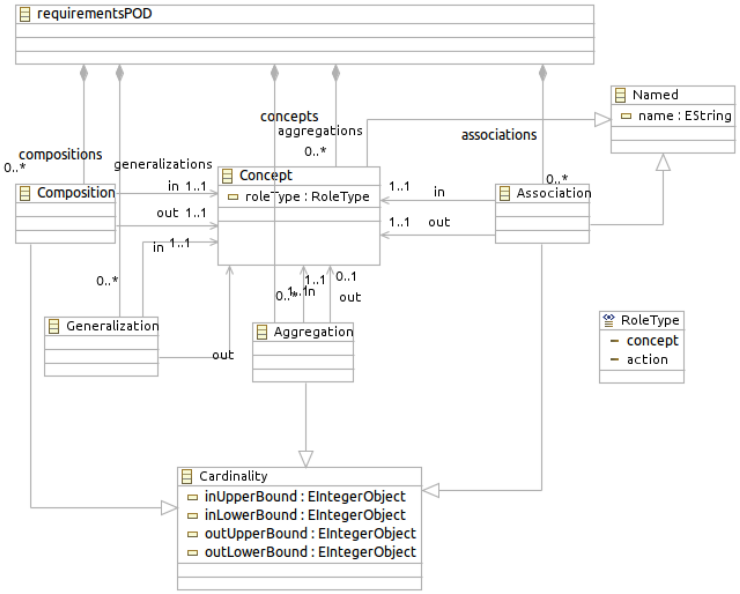

FIGURA 4.3: Metamodelo para el Diagrama de Ontologías del Dominio.

Los diagramas que surgen de los metamodelos antes mostrados y la metáfora gráfica que adopta cada concepto en los diagramas son presentados, ejemplo de por medio, en las secciones 4.4.2 y 4.4.3.

4.3. Adaptación del Metamodelo CRIO

Como se ha descripto en la Sección 2.6, el metamodelo CRIO es la base fundamental de ASPECS. En esta sección nos centraremos en la primera fase de dicha metodología, la fase *Requerimientos del Sistema*.

Para dar soporte a la manipulación informática de los modelos fue necesario adaptar CRIO, presentado en la sección 2.5, para su implementación en EMP. Dado que el metamodelo teórico es demasiado abstracto para ser implementado directamente fue necesario agregar ciertos elementos que hicieron posible su implementación completa. En la Figura 4.4 se muestra una imagen del modelo de EMF para CRIO y en las siguientes subsecciones se describirá en detalle cada una de las vistas que surgen de dicha adaptación.

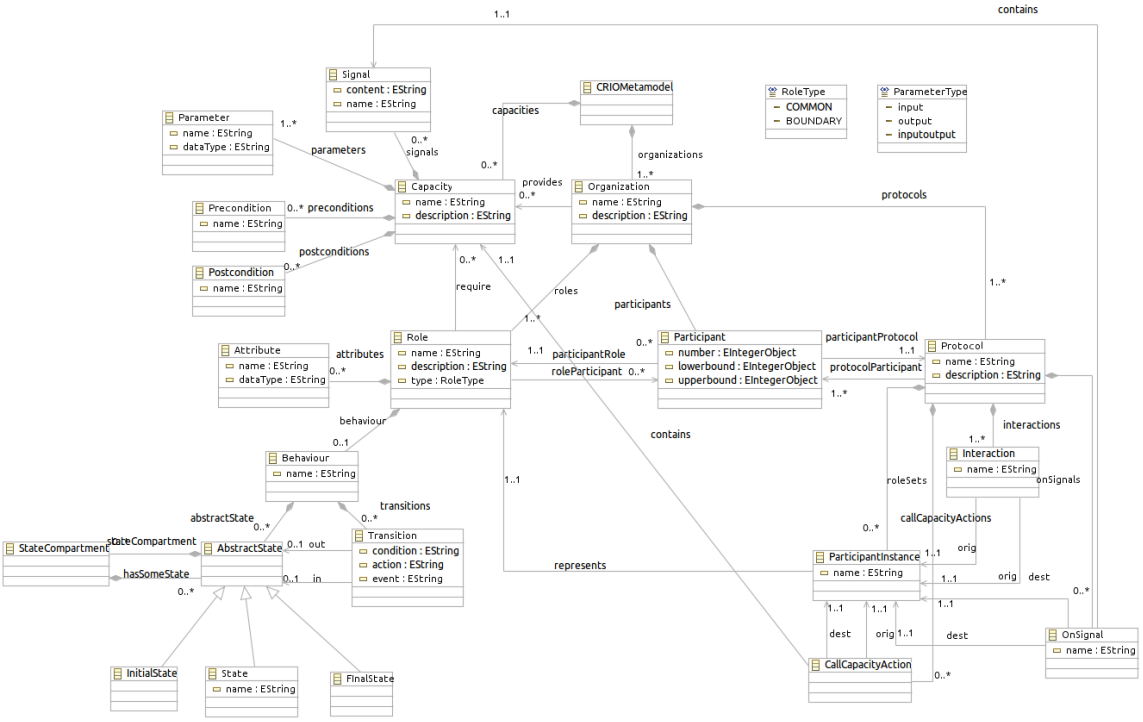

FIGURA 4.4: Metamodelo CRIO representado en el Ecore de EMF.

CUADRO 4.2: Metamodelo del Dominio del Problema y sus Vistas

	Social	Individual
Estático	*Vista Organizacional*	*Vista Organizacional (Rol)*
Dinámico	*Vista Interacción*	*Vista de Comportamiento*

4.3.1. Vista Organizacional

El objetivo de la vista organizacional es identificar por cada requerimiento un comportamiento global expresado en una organización. En otras palabras, permite capturar los aspectos estáticos y estructurales del enfoque organizacional. Esta vista toma como entrada lo realizado en la actividad *Descripción de Requerimientos del Dominio* y a partir del análisis de los Casos de Uso y de la estructura de la ontología, identifica un conjunto de organizaciones cada una de las cuales estará asociada con al menos un Caso de Uso.

En nuestra adaptación del metamodelo CRIO, *CRIOMetamodel* es el elemento raíz del *Diagrama Organizacional*. El mismo está compuesto de los conceptos *Organization* y *Capacity*. El concepto *Organization* está definido por los siguientes atributos: (i) *name*, es obligatorio que todas las organizaciones contengan un nombre, y opcionalmente, (ii) *description* que es un breve descripción del propósito de la organización. A su vez,

Organization está compuesto por los conceptos de *Role*, *Protocol* y *Participant*, que serán detallados a continuación.

En *Role*, tanto *name* como *description* tienen el mismo significado que en *Organization*; descripto en el párrafo anterior. Adicionalmente, posee una enumeración como atributo denominado *RoleType*, que puede tomar los valores *Common* o *Boundery* que serán explicados en la Sección 4.4.4. Además, *Role* contiene dos atributos adicionales que son diagramados como secciones insertos dentro del mismo concepto en el editor gráfico. El primero permitirá definir los atributos que representan los valores de entrada del *Diagrama de Comportamiento*. La segunda sección está reservada para definir el comportamiento ("Behaviour") del rol el cual será explicado más adelante, en las Sección 4.3.3. *Participant* es usado para relacionar roles y protocolos como así también para especificar el número de agentes que podrán tomar el rol. *Protocol* describe un escenario de interacción entre los roles participantes; esto se analizará más en detalle en la subsección 4.3.2.

El concepto de *Capacity* se refiere a las posibilidades que tiene el agente de completar un tarea o alcanzar un meta. En otras palabras, representa las competencias o habilidades que un agente debe tener para poder tomar un rol. Fue creado para promover la reusabilidad y modularidad y está compuesto por cuatro elementos, cada uno de los cuales representa una sección del concepto dentro del editor gráfico. La primera sección, *Parameters*, representa los valores de entrada necesarios para los servicios que implementa la capacidad. La segunda, permite definir las señales que simbolizan los eventos disparados por el agente que juega el rol notificando la finalización de una tarea. Estas señales poseen nombre y opcionalmente un colección de valores de retorno. Finalmente, las secciones referentes a las pre- y post- condiciones son restricciones lógicas definidas tanto para los valores de entradas como para los valores de salida, respectivamente.

4.3.2. Vista de Interacción

La vista de interacción permite capturar uno de los aspectos dinámicos del enfoque organizacional, más precisamente el de interacción entre roles. El reto de esta vista es describir una secuencia de intercambio de información entre los roles involucrados en cada escenario. Estos escenarios son deducidos desde un conjunto de descripciones textuales provenientes de los requerimientos iniciales, de los escenarios de uso del sistemas como así también de las salidas de las actividades de identificación de las organizaciones, sus roles y interacciones.

El concepto *Protocol* es el elemento raíz del *Diagrama de Interacción*. Los protocolos están compuestos por *Interaction*, *ParticipantInstance*, *CallCapacityAction* y *onSignal*.

En cuanto a *Interaction* debe existir al menos una instancia que vincule los roles participantes del diagrama siendo esta una restricción que en caso de no ser satisfecha invalidaría el modelo. Todas las interacciones deben tener un nombre y dos atributos: *orig* y *dest*, que representan respectivamente el rol origen como el rol destino de la interacción. El concepto de *ParticipantInstance* es el representante del rol en la Vista de Interacción, tiene dos atributos, por un lado, *name* del tipo *EString* y por el otro, *represents* que permite asociar este concepto con *Role*. *ParticipantInstance* fue creado dado que ningún concepto en el metamodelo, en este caso *Role*, puede ser usado en dos vistas diferentes incluso si tienen metáforas gráficas distintas en cada diagrama.

CallCapacityAction representa un auto-mensaje asincrónico. Posee un atributo del tipo *Capacity* llamado *contains*. Este concepto representa la posibilidad del rol de invocar e interactuar con el agente que lo juega. El último concepto, *onSignal*, representa los eventos disparados por el agente notificando la finalización de una tarea que realizaba. Uno de sus atributos es *contains* del tipo *Signal*. Tanto *CallCapacityAction* y *onSignal* son auto-mensajes del rol. Por esta razón, existe una restricción o regla que verifica que el rol origen y el rol destino sean el mismo.

4.3.3. Vista de Comportamiento

Como hemos definido con anterioridad, el concepto de rol contribuye en alcanzar los requerimientos de la organizacion dentro del cual está definido. Es por ello que se ha definido la *Vista de Comportamiento* que permite representar la dinámica de las instancias. Es decir, se describe el comportamiento de un rol en términos de estados y transiciones. En CRIO, tal comportamiento es visualizado en un diagrama de Transición-Estados (Statechart)[58]. Janeiro permite la modelización de Transición-Estados por cada rol existente. Sin embargo, versiones futuras de la aplicación permitirán integrar otras representaciones tales como son las Redes de Petri, los Diagramas de Actividad, etc.

El elemento raíz definido para este editor es el concepto de *Behaviour*, que a su vez está compuesto por dos conceptos que son los elementos básicos de un diagrama de transición-estados: *AbstractState* y *Transition*. El primero representa una generalización de los estados en los que se puede encontrar un rol en un momento determinado. Estos estados a saber son: estado inicial, toda máquina tiene al menos un estado inicial; estados regulares, que representan los distintos estados definido para ese rol; y por último, un estado final que puede o no estar presente en un diagrama siempre y cuando el diseñador lo considere necesario.

Este diagrama, a través *StateCompartment*, permite que sea posible representar estados anidados en el diagrama.

4.4. Utilización de Janeiro Studio en ASPECS

A fin de ilustrar el uso de la herramienta por los diseñadores del sistema, presentaremos en este capítulo un caso de estudio desarrollado previamente por miembros del equipo.

4.4.1. Caso de Estudio

El caso de estudio que se detallará en esta sección fue presentado originalmente en [1]. El mismo describe el funcionamiento de una planta industrial para la fabricación de automóviles ubicado en el Este de Francia. La misma abarca más de 250 hectáreas pareciendo, por la cantidad de edificios y el tránsito interno de vehículos, a un pequeño pueblo. Además, esta planta se encuentra en constante evolución producto de la creciente demanda de automóviles a nivel mundial. La plantilla de personal es de aproximadamente 19000 personas en total divididas en tres turnos para cubrir las 24 horas por día de funcionamiento. El movimiento de camiones, que entran y salen del área, es en promedio de 1600 por día. Finalmente, resulta difícil que la fábrica crezca en nuevos edificios por las características geográficas del entorno, lo que fuerza al rediseño constante de la infraestructura.

Cada edificio posee una línea de producción e intercambia con otros sus productos usando camiones que realizan un recorrido predefinido. El intercambio de productos entre estos edificios se denomina Cluster Building. Identificar estos clusters es de una gran utilidad cuando se realiza un nuevo trazado de rutas o una reubicación de los edificios existentes. Llevar a cabo modificaciones puede provocar el reposicionamiento de los diferentes talleres de trabajo requeridos para el plan de procesamiento de materiales dentro de los edificios. Cada taller usualmente recibe materiales desde el exterior (materias primas) y puede, además, enviar sus subproductos a otros talleres. Por lo que una modificación en las ubicaciones de los talleres podría generar perturbaciones en el flujo del tráfico interrumpiendo el funcionamiento normal de la planta. Esta situación deja entrever que una adecuada posición de los talleres en los edificios existentes es un potencial trabajo de optimización que puede resultar en un incremento de la producción en la planta.

4.4.2. Diagrama de Requerimientos del Dominio

Para que el desarrollo de un sistema sea exitoso, el mismo debe contar con una correcta documentación de sus requerimientos tanto funcionales como no funcionales. En otras palabras, el objetivos principal de la fase de requerimientos es capturar las necesidades o intereses de las personas que serán afectadas por el desarrollo del sistemas además de contar con una descripción global del comportamiento de la aplicación objeto de desarrollo. La metodología ASPECS define para tal objetivo el diagrama *Descripción de los Requerimientos del Dominio* (Figura 4.5); inspirado en el Diagrama de Casos de Uso de UML permitiendo representar las descripciones funcionales y no funcionales utilizando un lenguaje específico del dominio provisto por el usuario.

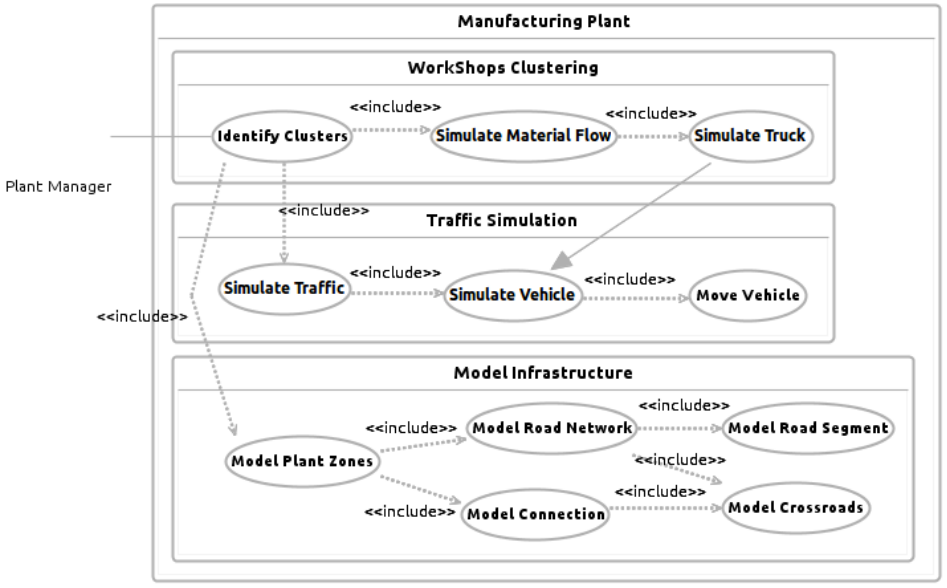

FIGURA 4.5: Diagrama de Requerimientos del Dominio

4.4.3. Diagrama de Ontología del Problema

Los requerimientos de los usuarios en su mayoría son expresados utilizando el lenguaje natural. A su vez, estos requerimientos tienen implícitos términos propios del dominio de aplicación que son extraídos de los casos de usos definidos en el *Diagrama de Requerimientos del Dominio*. El *Diagrama de Ontología del Dominio* (DOD)(Figura 4.6) permite identificar, modelados como un diagrama de clases, los conceptos, predicados y acciones relevantes para el dominio del problema.

En el estudio de caso se definió una serie de conceptos y acciones que son propios de los caminos que conectan los distintos talleres de la planta industrial. Por ejemplo, el segmento ("RoadSegment") está conformado por carriles ("RoadLane") donde circulan

(acción "Move") los distintos vehículos ("Vehicles") que pueden ser autos o camiones. Como se puede advertir estos son algunos de los conceptos necesarios para establecer un entendimiento común y compartido del dominio entre los diseñadores.

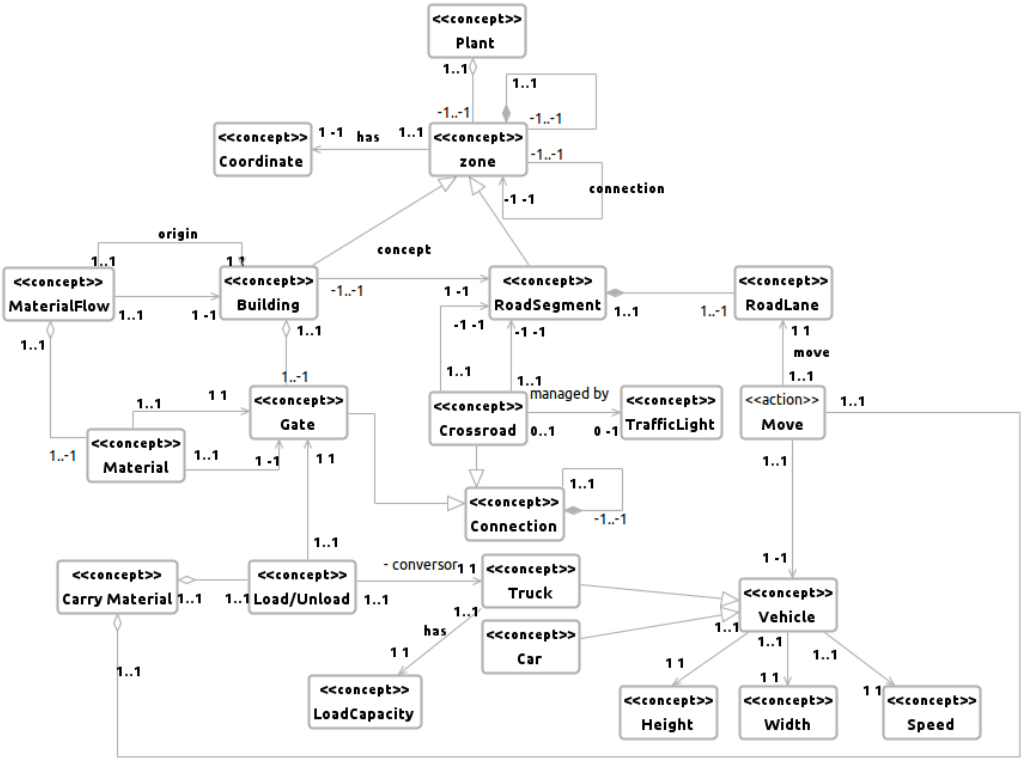

FIGURA 4.6: Diagrama de Ontología de Problema

4.4.4. Diagrama Organizacional

El principal diagrama del metamodelo CRIO es el *Diagrama Organizacional*. Creado para capturar los aspectos estáticos y estructurales del enfoque organizacional. El objetivo de este diagrama es representar los requerimientos del usuario en forma de organizaciones. Considere la Figura 4.7, cada organización es definida como un paquete con el estereotipo "≪organization≫" e inmediatamente más abajo una etiqueta con el nombre de la organización. El mismo contiene todos los roles que estén involucrados en la organización, los protocolos y las relaciones entre roles y protocolos además de proveer el contexto común.

De igual manera que las organizaciones, el concepto de rol está representado por un rectángulo con el estereotipo "≪role≫". La identificación de los roles surge de descomponer el comportamiento global de una organización en unidades más reducidas. Cada uno de estos fragmentos representarán los comportamientos exhibidos por los roles. Dos

tipos de roles son posibles identificar en una organización: (i) "≪COMMON≫", ubicado dentro del sistema y capaz de interactuar con otros roles del mismo tipo además del rol Boundery; (ii) "≪BOUNDERY≫", que representa aquellos elementos que se encuentran ubicados en los límites o frontera entre el sistema y el exterior y es responsable de las interacciones que ocurren en ese límite. Cada rol, además, posee dos secciones dedicadas. La primera para los atributos que son utilizados como parámetros de entrada para el comportamientos del rol y la segunda para definir, mediante una Diagrama de Comportamiento, la dinámica de las instancias (indicado en la figura como "≪behavior≫").

Mediante el concepto de protocolo es posible descomponer los objetivos organizacionales; aunque con un nivel de abstracción mayor que el de los roles. Un protocolo de interacción y los roles asociados al mismo indican cual es la secuencia de mensajes necesaria para alcanzar uno de los objetivos definidos para la organización. El protocolo en el diagrama está representado por un círculo de color gris con una etiqueta afuera del mismo que indica el estereotipo "≪protocol≫" y otra donde es posible especificar su nombre. A su vez, la cardinalidad de esta asociación representa el número de instancias de agentes que podrán jugar este rol por instancia de organización. Cada uno de los protocolos es detallando por el *Diagrama de Interacción*.

Por último, las capacidades representan las competencias o habilidades que debe tener un agente para poder tomar un rol dentro de la organización. Una *Capacidad* es la descripción de un know-how o servicio. Estas capacidades están representadas por una clase estereotipada con una etiqueta "≪capacity≫" y son diagramadas fuera de la organización. La razón por la que la capacidad es puesta afuera, se debe a que fue concebida como una abstracción de alto nivel que promueve la reusabilidad y modularidad y que puede proveer sus características a diferentes organizaciones. Cuando una capacidad es requerida por un rol estas son enlazadas mediante un flecha etiquetada con "≪require≫".

El concepto de *Capacidad* posee una serie de secciones que permiten definir cuestiones relacionadas a la misma:

- **Parámetros:** Estos son parámetros de entrada de la capacidad y está conformada por su nombre y tipo de dato que representa.
- **Señales:** Los servicios que implementan una capacidad disparan señales dirigidas al rol para indicar la finalización o el estado de ejecución de alguna actividad. Y, potencialmente parámetros para comunicar los resultados de la ejecución.
- **Pre- y Post- condiciones**: Son restricciones lógicas definidas tanto para los valores de entradas como los de salida, respectivamente.

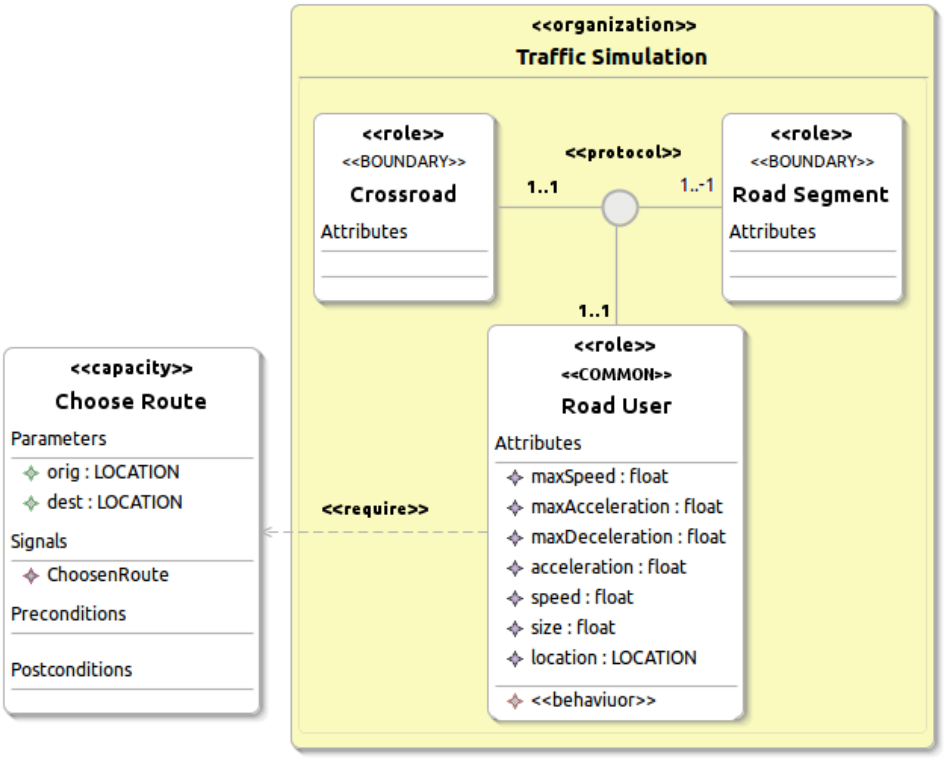

FIGURA 4.7: Diagrama Organizacional.

Centraremos nuestra atención en el organización *Traffic Simulation* de nuestro caso de estudio. Como se puede observar en la Figura 4.7, fueron identificados tres roles: *Road User*, *Crossroad* y *RoadSegment*. El primer rol es definido como *Common Role* mientras que los otros dos, al estar ubicados en los límites entre la planta industrial y la región exterior, son definidos como *Boundery Role*; queda a decisión del diseñador considerarlos de una u otra manera. El funcionamiento de la organización es la siguiente: un conductor (RoadUser) puede conducir su vehículo a través del camino y a través de los cruces (Crossroad) siempre que las condiciones del ambientes lo permiten (Por ejemplo, ningún otro conductor circula por un cruce al mismo tiempo).

4.4.5. Diagrama de Interacción

Permite capturar, a través de la descripción de un escenario, unos de los aspectos dinámicas del enfoque organizacional. La finalidad del *Diagrama de Interacción* (figura 4.8) es describir la secuencia de intercambio de información entre roles que tiene lugar en un protocolo. El diagrama, que describe a un protocolo del *Diagrama Organizacional*, está compuesto por representantes, mensajes o interacciones, llamados a capacidades y señales.

Gráficamente un participante consta de un "RoleHead"; que es un rectángulo con una etiqueta "≪participant≫". Además, posee una línea vertical punteada que representa la línea de vida del rol que corresponde a su función "live" y es dónde se originan o es receptor de los mensajes como así también de los llamados a capacidad o señales. En el "RoleHead" también es posible establecer el nombre de instancia además del nombre del rol; utilizando el formato "nombreInstancia:NombreRol", dónde "nombreInstancia" define el/los destinatario(s) o receptor(es) de los mensajes que llegan al participante y "NombreRol" representa el rol definido en el Diagrama Organizacional. Tres tipos de mensajes son posible en el diagrama: Si el mensaje es de tipo broadcast, denotado por "*" como nombre de instancia, el mensaje tiene como destinatario a todos los agentes que juegan el rol; en cambio, si el mensaje está dirigido a un destinatario que es elegido al azar por la plataforma se utilizará el símbolo "?"; por último, si el mensaje está dirigido a un *roleplayer* en particular basta poner la dirección del agente ("AgentAddress") o nombre.

Los intercambios de información entre los roles están representados por un línea dirigida con una etiqueta indicativa del nombre de la interacción además de los parámetros asociados a dicho mensaje. Todos los mensajes se suponen que son siempre asincrónicos dado que no es posible establecer a priori cuando puede el receptor del mensaje responder.

Los roles tienen asociados tareas u operaciones, además de los llamados a capacidades, que representan su comportamiento esperado dentro de la organización. Dentro de estas operaciones existen dos tipos especiales denominados "call" y "onSignal" que son las formas en que el rol interactúa con su agente. La primera operación invoca su Know-How para la realización de una actividad. Estos llamados a capacidad deben proveer resultados que son provistos por las señales. Una señal es disparada por el agente indicado la finalización de una tarea o informando sobre el estado de ejecución de una actividad. Además, y al igual que las interacciones, tanto "call" como "onSignal" son también mensajes del tipo asincrónicos que previenen, por ejemplo, que el rol se bloquee cuando realiza un llamado a una capacidad y deba esperar por un resultado. Con esta característica, en cambio, el rol puede continuar ejecutando otras actividades que tenga definidas.

El diagrama de interacción descripto para la organización *Traffic Simulation* se muestra en la Figura 4.8. La misma ilustra a un *Road User* solicitando un permiso para circular por un *Crossroad*. Una vez recibida la solicitud, *Crossroad* verifica si el pedido cumple con las reglas de tráfico local (giros obligatorios, etc) para después solicitar información de tráfico al próximo *Road Segment*. Por lo tanto, si es posible acceder al nuevo *Road*

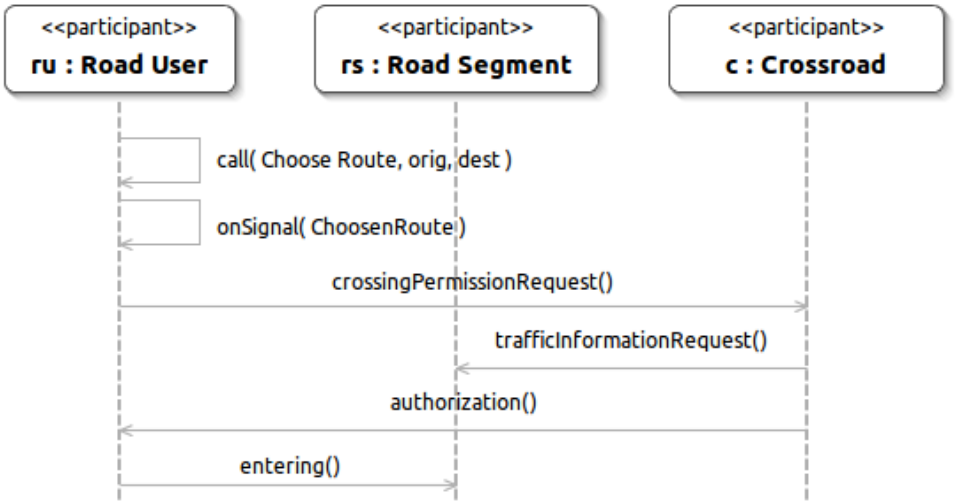

FIGURA 4.8: Diagrama de Interacción.

Segment entonces *Crossroad* otorga los permisos a *Road User* para que pueda circular por el camino solicitado.

4.4.6. Diagrama de Comportamiento

En la parte inferior del rol en el diagrama organizacional se puede observar una sección referente a su comportamiento, denominado "≪behavior≫". Los estados de un rol se refieren al conjunto de valores de sus atributos. Este comportamiento está representado por un *Diagrama de Transición-Estados*. El diagrama está compuesto básicamente por estados y transiciones entre los estados. En la paleta del diagrama se pueden encontrar tres tipos de estados: los pseudo-estados Inicial y Final, y el Estado Regular. La metáfora gráfica asociada para cada uno de ellos esta compuesta de: el estado inicial, representado por un circulo negro, el estado final por un circulo dentro de otro también de color negro y los estados regulares como una caja rectangular.

Las transiciones son las relaciones entre los estados y están representadas gráficamente por flechas dirigidas con una etiqueta. La etiqueta mencionada consta de tres partes "evento/condición/acción".

En nuestro ejemplo describiremos el comportamiento del rol *Road User* (Figura 4.8). Por defecto y por la misma restricción del diagrama, todos comienzan con un estado "idle". Una vez que se ha tomado la decisión de cual será el camino, él vehículo empieza a recorrer la calle. El rol permanecerá en este estado hasta alcanzar un cruce. Cuando se ha alcanzado un cruce, el rol enviará el mensaje *requestCrossingPersmission* y aguardará por un respuesta. Por último, pueden ocurrir dos situaciones: la primera, el rol recibe una respuesta (*crossingPermission*) y continúa camino por una de las calles

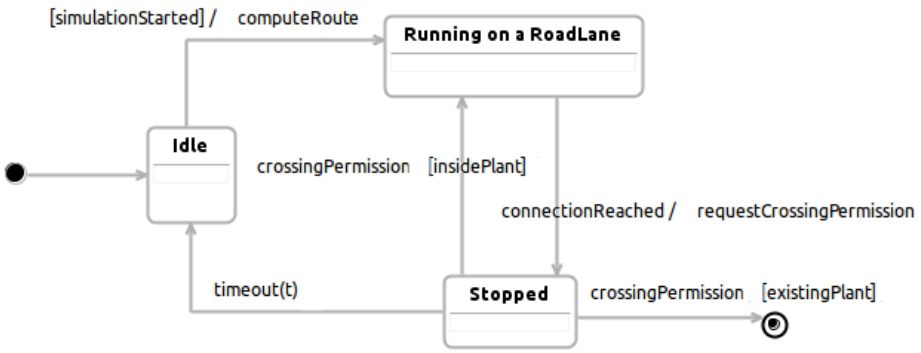

FIGURA 4.9: Diagrama de Comportamiento.

prevista en el ruta o sale de la planta. La segunda situación es que el rol no reciba ninguna respuesta antes que expire la ventana de tiempo que tiene asignadas la espera de respuestas por lo que debe calcular una nueva ruta.

4.5. Conclusiones

En el presente capítulo se describe el primer prototipo de la herramienta de desarrollo *Janeiro Studio*. Para alcanzar este objetivo fue necesario una adaptación del metamodelo CRIO a la tecnología EMF de Eclipse que concluyeron en la definición de los 3 Ecores que actualmente posee la herramienta. El *Diagrama de Descripción del Dominio* y el *Diagrama de Ontología* son soportados cada uno por un Ecore, mientras que el *Diagrama Organizacional*, de *Interacción* y *Comportamiento* son "vistas" que surgen del Ecore restante. Estos cinco diagramas en su conjunto, permiten abordar la primera de las tres fases del proceso de desarrollo de ASPECS; denominada *Dominio del Problema*.

Para probar la validez de la herramienta fue desarrollado un ejemplo que modela una simulación del tráfico de vehículos de la planta industrial de la marca Peugeot-PSA. Los requerimientos iniciales fueron plasmados en el *Diagrama de Requerimientos del Dominio* y el *Diagrama de Ontología*. A continuación se modelizó la organización principal (*Traffic Simulation*) y los *Diagramas de Interacción y de Comportamiento* que describen la dinámica del sistema.

Capítulo 5

Conclusiones y Trabajos Futuros

El presente Trabajo Final Integrador muestra los avances realizados en Janeiro Studio. El mismo brinda soporte a la primera fase de la metodología ASPECS. Este trabajo se enmarca en el desarrollo de Conceptos necesarios para el análisis, diseño e implementación de Sistemas Multiagentes utilizando un enfoque Organizacional. En este contexto el equipo ha desarrollo diferentes aspectos teóricos tales como una Metodología (ASPECS), Metamodelos necesarios (CRIO) y lenguaje de programación (SARL/JANUS). Por lo tanto, este trabajo busca vincular estos aspectos y brindar soporte informático al desarrollo de un SMA mediante una herramienta CASE.

Para esta primera etapa fueron desarrollados cinco diagramas a saber: *Diagrama de Requerimientos del Dominio* útil para capturar las necesidades de los usuarios. El *Diagrama de Ontología* describe los términos usados en el lenguaje específico del dominio de aplicación. El *Diagrama Organizacional* es en donde se definen las organizaciones y dentro de los mismo la colección de roles, protocolos e interacciones. El *Diagrama de Interacción*, donde se representa el intercambio de mensajes y la secuencia asociados a los mismos. Por último, el *Diagrama de Statechart* que permite representar los estados posibles en los que puede estar un rol.

A su vez presenta la infraestructura y entorno de desarrollo establecidos para la implementación de Janeiro. El mismo describe una estructura genérica fácilmente reutilizable para otros proyectos de envergadura. Cabe destacar que este entorno ha sido adoptado en el Grupo de Investigación en Tecnologías Informáticas Avanzadas (GITIA) de la Facultad Regional Tucumán de la UTN y está siendo actualmente utilizado para el desarrollo de prototipos en otros proyectos de investigación. Entre los elementos que conforman la infraestructura señalada se encuentran: el sistema de versionado Git que permite la creación de redes de confianza entre desarrolladores de forma que los más

experimentados puedan validar las modificaciones previa inclusión al repositorio primario. Por otro lado, el servidor de integración continua facilita la rápida detección de errores reduciendo el impacto de los mismo en el futuro del proyecto. Esto es de vital importancia cuando muchos individuos participan el proyecto. Por último, el uso de Maven simplifica la gestión de las dependencias y la generación de un entorno unificado y homogéneo de compilación.

En cuanto a desarrollo se refiere se ha utilizado la suite de Eclipse, entre ellas tenemos a RCP que simplifica la creación de aplicaciones basado fuertemente en la idea de reutilización. EMF, utilizado para la especificación de DSL y en el cual se ha realizado una adaptación del metamodelo teórico CRIO. A su vez, la adopción de GMF nos permitió definir, en primer término, los Editores y Vistas que la aplicación tendrá como así también la metáfora gráfica de los conceptos de los metamodelos. Esta suite ha demostrado a lo largo de los años ser una de las mejores debido a su fácil integración con diferentes componentes y constante evolución para el desarrollo de aplicación con altas prestaciones y complejidad.

En lo que se refiere a los trabajos futuros, el próximo paso en la evolución de Janeiro está previsto realizar los modelos para el dominio de agencia. Los mismo requiere de la definición de un conjunto de reglas de transformación que nos permita generar los elementos necesarios a partir de los modelos creados para el “Dominio del Problema”. A su vez, se encuentra en desarrollo un módulo que permitirá verificar y validar los modelos diseñados.

Como hemos mencionado anteriormente, nuestra intención con Janiero Studio es brindar al diseñador de sistemas de una herramienta que provea el mayor soporte posible a la metodología ASPECS. Además, está previsto avanzar en los módulos que permiten la generación de código, los procesos de debbuging y la integración con la plataforma Janus. Logrando así contribuir en el armado de un ecosistema para abordar sistemas complejos mediante la utilización modelos basados en sistemas multiagentes holónicos.

Por otro lado, el equipo de trabajo a desarrollado un lenguaje de programación orientado a agente denominado SARL [59]. El mismo cuenta con conceptos similares a los de CRIO. A su vez, el compilador genera, previa a la generación del código definitivo, un modelo EMF del Sistema Multiagente. Este enfoque permitiría reutilizar el modelo EMF de SARL como el “Modelo de la Solución” en ASPECS. Así Janeiro podría, mediante transformaciones de modelos, cubrir el ciclo completo desde el análisis, concepción e implementación de un Sistema Multiagente basado en ASPECS.

Por último, en cuanto a la evolución del entorno de desarrollo propuesto, se evalúa la integración de herramientas para el soporte de enfoques como “Continous Deployment” y, posteriormente, “Continous Delivery”.

Bibliografía

[1] Massimo Cossentino, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abder Koukam. Aspecs: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20(2):260 – 304, 2010.

[2] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, Chichester, U.K, 2nd edition edition, 2009. ISBN 9780470519462.

[3] Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In Yves Demazeau, Edmund H. Durfee, and Nicholas R. Jennings, editors, *ICMAS'98*, july 1998.

[4] Andrea Omicini. Soda: Societies and infrastructures in the analysis and design of agent-based systems. In *In this volume*, pages 185–193. Springer-Verlag, 2000.

[5] Virginia Dignum, Javier Vázquez-salceda, and Frank Dignum. Omni: Introducing social structure, norms and ontologies into agent organizations. In *in 'PROMAS*, pages 181–198. Springer, 2004.

[6] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology, 2003.

[7] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Tropos: An agent-oriented software development methodology. 2003.

[8] M. Cossentino and C. Potts. Passi: a process for specifying and implementing multi-agent systems using uml, 2002.

[9] James Odell, Marian Nodine, and Renato Levy. A metamodel for agents, roles, and groups. In James Odell, P. Giorgini, and Jörg Müller, editors, *Agent-Oriented Software Engineering (AOSE) IV*, Lecture Notes on Computer Science. Springer, 2005.

[10] P. Marcenac. Modélisation de systémes complexes par agents. *Techniques et Sciences Informatiques*, 16(8), 1997.

[11] Sebastian Rodriguez, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abder Koukam. An analysis and design concept for self-organization in holonic multi-agent systems. In *Proceedings of the 4th international conference on Engineering self-organising systems*, ESOA'06, pages 15–27, Berlin, Heidelberg, 2007. Springer-Verlag.

[12] Matthieu Amiguet. *MOCA: un modele componentiel dynamique pour les systemes multi-agents organisationnels*. PhD thesis, Université de Neuchetel, 2003.

[13] Sebastian Rodriguez, Vincent Hilaire, and Abder Koukam. Towards a methodological framework for holonic multi-agent systems. In *Fourth International Workshop of Engineering Societies in the Agents World*, pages 179–185, Imperial College London, UK (EU), 29-31 October 2003.

[14] Sebastian Rodriguez, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abderrafiâa Koukam. An analysis and design concept for self-organization in holonic multi-agent systems. In Sven A. Brueckner, Salima Hassas, Márk Jelasity, and Daniel Yamins, editors, *Engineering Self-Organising Systems*, number 4335 in Lecture Notes in Computer Science, pages 15–27. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-69867-8, 978-3-540-69868-5. URL http://link.springer.com/chapter/10.1007/978-3-540-69868-5_2.

[15] Massimo Cossentino, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abderrafiâa Koukam. Aspecs: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20(2):260–304, 2010. ISSN 1387-2532, 1573-7454. doi: 10.1007/s10458-009-9099-4. URL http://link.springer.com/article/10.1007/s10458-009-9099-4.

[16] Gerhard Weiss. *Multiagent Systems*. The MIT Press, Cambridge, Massachusetts, second edition edition edition, 2013. ISBN 9780262018890.

[17] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: the gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3), July 2003.

[18] Francisco Leal and João Rodriguez. Message: Methodology for Engineering Systems of Software Agents. Technical report, Telecom Italia Lab and PT Inova
ccãl, 2001.

[19] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, JörgP.

Müller, and James Odell, editors, *Agent-Oriented Software Engineering IV*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-20826-6.

[20] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 3rd International Conference on Multi Agent Systems*, ICMAS '98, pages 128–, Washington, DC, USA, 1998. IEEE Computer Society.

[21] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *Agent-Oriented Software Engineering IV*, number 2935 in Lecture Notes in Computer Science, pages 214–230. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-20826-6, 978-3-540-24620-6. URL http://link.springer.com/chapter/10.1007/978-3-540-24620-6_15.

[22] UML. Unified Modeling Language (UML) Superstructure V 2.4.1, au 2011. Version 2.4.1.

[23] Scott A DeLoach and Juan Carlos Garcia-Ojeda. O-mase: a customisable approach to designing and building complex, adaptive multi-agent systems. *International Journal of Agent-Oriented Software Engineering*, 4(3):244–280, 2010.

[24] Scott A. DeLoach. The mase methodology. In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, number 11 in Multiagent Systems, Artificial Societies, and Simulated Organizations, pages 107–125. Springer US, 2004. ISBN 978-1-4020-8057-9, 978-1-4020-8058-6. URL http://link.springer.com/chapter/10.1007/1-4020-8058-1_8.

[25] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000. ISSN 1387-2532, 1573-7454. doi: 10.1023/A:1010071910869. URL http://link.springer.com/article/10.1023/A%3A1010071910869.

[26] Luca Cernuzzi, Ambra Molesini, and Andrea Omicini. The gaia methodology process. In *Handbook on Agent-Oriented Design Processes*, pages 141–172. Springer, 2014.

[27] Juan Pavón and Jorge Gómez-Sanz. Agent oriented software engineering with ingenias. In Vladimír Mařík, Michal Pěchouček, and Jörg Müller, editors, *Multi-Agent Systems and Applications III*, number 2691 in Lecture Notes in Computer Science, pages 394–403. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-40450-7, 978-3-540-45023-8. URL http://link.springer.com/chapter/10.1007/3-540-45023-8_38.

[28] Giovanni Caire, Wim Coulier, Francisco J. Garijo, Jorge Gomez, Juan Pavon Mestras, Francisco Leal, Paulo Chainho, Paul E. Kearney, Jamie Stark, Richard Evans, and Philippe Massonet. Agent oriented analysis using message/uml. In Michael Wooldridge, Gerhard Weiß, and Paolo Ciancarini, editors, *Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001, Revised Papers and Invited Contributions*, volume 2222 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2002. ISBN 3-540-43282-5.

[29] Lin Padgham and Michael Winikoff. Prometheus: A methodology for developing intelligent agents. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III*, number 2585 in Lecture Notes in Computer Science, pages 174–185. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-00713-5, 978-3-540-36540-2. URL http://link.springer.com/chapter/10.1007/3-540-36540-0_14.

[30] Thomas Juan, Adrian Pearce, and Leon Sterling. Roadmap: extending the gaia methodology for complex open systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 3–10. ACM, 2002.

[31] Ariel Fuxman, Lin Liu, John Mylopoulos, Marco Pistore, Marco Roveri, and Paolo Traverso. Specifying and analyzing early requirements in tropos. *Requirements Engineering*, 9(2):132–150, 2004. ISSN 0947-3602, 1432-010X. doi: 10.1007/s00766-004-0191-7. URL http://link.springer.com/article/10.1007/s00766-004-0191-7.

[32] Mirko Morandini, Fabiano Dalpiaz, Cu Duy Nguyen, and Alberto Siena. The tropos software engineering methodology. In *Handbook on Agent-Oriented Design Processes*, pages 463–490. Springer, 2014.

[33] Massimo Cossentino. From requirements to code with the passi methodology. *Agent-oriented methodologies*, 3690:79–106, 2005.

[34] Carole Bernon, Valérie Camps, Marie-Pierre Gleizes, and Gauthier Picard. Engineering adaptive multi-agent systems: The adelfe methodology. *Agent-Oriented Methodologies*, pages 172–202, 2005.

[35] David Isern, David Sánchez, and Antonio Moreno. Organizational structures supported by agent-oriented methodologies. *J. Syst. Softw.*, 84(2):169–184, 2011. ISSN 0164-1212. doi: 10.1016/j.jss.2010.09.005. URL http://dx.doi.org/10.1016/j.jss.2010.09.005.

[36] Juan C. Garcia-Ojeda, Scott A. DeLoach, and Robby. agentTool III: from process definition to code generation. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1393–1394, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[37] JuanC. Garcia-Ojeda, ScottA. DeLoach, Robby, WalamitienH. Oyenan, and Jorge Valenzuela. O-mase: A customizable approach to developing multiagent development processes. In Michael Luck and Lin Padgham, editors, *Agent-Oriented Software Engineering VIII*, volume 4951 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-79487-5.

[38] Luca Cernuzzi and Franco Zambonelli. Gaia4e: A tool supporting the design of mas using gaia.

[39] Michael Wooldridge, NicholasR. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000. ISSN 1387-2532.

[40] Juan Pavón and Jorge Gómez-Sanz. Agent oriented software engineering with ingenias. In Vladimír Mařík, Michal Pěchouček, and Jörg Müller, editors, *Multi-Agent Systems and Applications III*, volume 2691 of *Lecture Notes in Computer Science*, pages 394–403. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-40450-7.

[41] Jorge J. Gomez-Sanz, Rubén Fuentes, Juan Pavón, and Ivan García-Magariño. Ingenias development kit: a visual multi-agent system development environment. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, AAMAS '08, pages 1675–1676, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[42] Lin Padgham and Michael Winikoff. Prometheus: A methodology for developing intelligent agents. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Agent-Oriented Software Engineering III*, volume 2585 of *Lecture Notes in Computer Science*, pages 174–185. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-00713-5.

[43] John Thangarajah, Lin Padgham, and Michael Winikoff. Prometheus design tool. In *Proceedings of the fourth international joint conference on Autonomous agents*

and multiagent systems, AAMAS '05, pages 127–128, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0.

[44] Roberto Caico, Massimo Cossentino, Luca Sabatucci, Valeria Seidita, and Salvatore Gaglio. Metameth: a tool for process definition and execution. In *Proceedings of the 7th WOA 2006 Workshop, From Objects to Agents (Dagli Oggetti Agli Agenti), Catania, Italy, September 26-27, 2006.*, 2006. URL http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-204/D06.pdf.

[45] ANTONIO Chella, Massimo Cossentino, and Luca Sabatucci. Tools and patterns in designing multi-agent systems with passi. *WSEAS Transactions on Communications*, 3(1):352–358, 2004.

[46] Vincent Hilaire. *Vers une approche de spécification, de prototypage et de vérification de Systèmes Multi-Agents.* Phd thesis, Université de Technologie de Belfort-Montbéliard, 2010.

[47] Sebastian Rodriguez. *From analysis to design of holonic multi-agent systems: A framework, methodological guidelines and applications.* PhD thesis, Universitè de Technologie de Belfort-Montbèliard and Universitè de Franche-Comptè, 2005.

[48] Nicolas Gaud. *Holonic Multi-Agent Systems: From the analysis to the implementation. Metamodel, Methodology and Multilevel simulation.* PhD thesis, Université de Technologie de Belfort-Montbéliard, Belfort. France, 2007.

[49] Juan P. Gruer, Vincent Hilaire, Abder Koukam, and P. Rovarini. Heterogeneous formal specification based on Object-Z and statecharts: semantics and verification. *Journal of Systems and Software*, 70(1-2):95–105, feb, 2004. doi: 10.1016/s0164-1212(02)00161-9. URL http://dx.doi.org/10.1016/S0164-1212(02)00161-9.

[50] Roger Duke, Gordon Rose, and Graeme Smith. Object-z: a specification language advocated for the description of standards. *COMPUTER STANDARDS AND INTERFACES*, 17, 1995.

[51] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, jun, 1987. ISSN 0167-6423. doi: 10.1016/0167-6423(87)90035-9. URL http://dx.doi.org/10.1016/0167-6423(87)90035-9.

[52] Nicolas Gaud, Vincent Hilaire, Stéphane Gall, Abderrafiâa Koukam, and Massimo Cossentino. A verification by abstraction framework for organizational multi-agent systems, 2008.

[53] Stéphane Galland, Nicolas Gaud, Sebastian Rodriguez, and Vincent Hilaire. Janus: Another yet general-purpose multiagent platform. In *7th Agent-Oriented Software Engineering Technical Forum (TFGAOSE-10)*. Agent Technical Fora, 2010.

[54] Paul Duvall, Stephen M. Matyas III, and Andrew Glover. *Continuous Integration: Improving Software Quality And Reducing Risk*. Addison-Wesley, 2007. ISBN 0321336380.

[55] Jeff McAffer, Jean-Michel Lemieux, and Chris Aniszczyk. *Eclipse Rich Client Platform*. The Eclipse Series. Pearson Education, 2 edition, may, 2010. ISBN 0321603788.

[56] Ian Sommerville. *Software Engineering*. Addison-Wesley, 9 edition, mar, 2010. ISBN 0137035152.

[57] Sebastián Rodríguez Pedro Araujo. Janeiro studio. Córdoba, Argentina, 2013.

[58] David Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987. ISSN 0167-6423. doi: 10.1016/0167-6423(87)90035-9. URL http://www.sciencedirect.com/science/article/pii/0167642387900359.

[59] Sebastian Rodriguez, Nicolas Gaud, and Stéphane Galland. Sarl: a general-purpose agent-oriented programming language. Warsaw, Poland, 2014. IEEE Computer Society Press.