RE4GAIA: UNA APROXIMACIÓN DE MODELADO DE REQUISITOS PARA EL DESARROLLO DE SISTEMAS MULTI-AGENTE

Departamento de Sistemas Informáticos y Computación

Universidad Politécnica de Valencia

Autor: David Blanes Domínguez

Director: Dr. Emilio Insfrán Pelozo

PARA LA OBTENCIÓN DEL TÍTULO DE MÁSTER

DE POSGRADO OFICIAL EN INGENIERÍA DEL SOFTWARE,

MÉTODOS FORMALES Y SISTEMAS DE INFORMACIÓN

Valencia, España

DICIEMBRE 2009

Fecha: **2 de diciembre de 2009**

Autor:	**David Blanes Domínguez**
Director:	**Dr. Emilio Insfrán Pelozo**
Título:	**RE4Gaia: Una aproximación de modelado de requisitos para el desarrollo de Sistemas Multi-Agente**
Departamento:	**Sistemas Informáticos y Computación**
Universidad:	**Universidad Politécnica de Valencia**
Mes:	**Diciembre**
Año:	**2009**

Resumen

Los Sistemas Multi-Agente (SMA) nacen como un nuevo paradigma de la Ingeniería del Software para diseñar y desarrollar sistemas de software complejos, adaptándose a las nuevas exigencias de éstos. Un SMA es un sistema compuesto por múltiples entidades de software autónomas que interactúan entre si para conseguir sus objetivos. A estas entidades se les llama *Agentes*. La investigación en la Ingeniería del Software (AOSE) en los últimos años se ha centrado en el desarrollo de metodologías para guiar el proceso de desarrollo de sistemas Multi-Agente. La metodologías AOSE, tratan de ofrecer un enfoque disciplinar para analizar, diseñar y desarrollar sistemas Multi-Agente usando métodos y técnicas específicas. Una de las metodologías más importantes es Gaia. La metodología Gaia explota las abstracciones organizacionales definiendo un proceso de diseño de sistemas Multi-Agente, proporcionando soporte a las fases de análisis y diseño. Sin embargo a pesar de su relevancia no proporciona ningún soporte explicito a la actividad de requisitos, una fase reconocida como crítica en el desarrollo del software.

En esta tesis de máster se propone resolver el problema de la falta de soporte a la fase de requisitos en Gaia proporcionando una aproximación metodológica: RE4Gaia. Esta aproximación se dividirá en dos fases. En la primera, el *Modelado de Requisitos,* se proveen los mecanismos necesarios para adquirir y modelar los requisitos del sistema. En la segunda, el *Proceso de Análisis de Requisitos*, se refinan los artefactos especificados previamente a un nivel de granularidad inferior que permite definir las correspondencias entre los requisitos especificados y los artefactos de la fase de análisis de Gaia. El Modelo de Requisitos propuesto se desarrollará dentro de un marco de Desarrollo de Software Dirigido por Modelos, de forma que se permitirá transformar de forma automática la especificación de requisitos funcionales a los modelos de análisis de Gaia.

Este trabajo parte de la realización de una Revisión Sistemática de Literatura con el objetivo de determinar *qué* técnicas de Ingeniería de Requisitos se usan en las metodologías Multi-Agente actuales y *cómo* se usan. Las conclusiones obtenidas de este estudio se tuvieron en cuenta para el posterior desarrollo de RE4Gaia. Se definieron los meta-modelos tanto para RE4Gaia como para el modelo de análisis de Gaia y las correspondientes transformaciones entre ellos. Se definió explícitamente el proceso de desarrollo, para la fase de requisitos, con el objetivo de guiar a los desarrolladores en la aplicación de este método, describiéndose los pasos a seguir y los artefactos a ser generados. Por último, se validó la propuesta mediante la resolución de un caso de estudio.

Abstract

The Multi-Agent Systems (MAS) has emerged as a new paradigm in the Software Engineering field to design and develop complex software systems, with the goal to adapt it to the new challenges in the software development. A MAS is a software system composed by multiple autonomous entities that interact each other to reach a goal. These entities are called *Agents.* The research in the Agent Software Engineering (AOSE) in the last years has been centered in the development of methodologies to guide the software process to build a MAS. The AOSE methodologies intend to offer a multi-disciplinary approach to analyze, design, and develop MAS using specific methods and techniques. One of these methodologies is Gaia. The Gaia methodology exploits the organizational abstraction to define a process to design MAS, giving support to the analysis and design phases. Despite the Gaia relevance, Gaia does not provide explicit support to the requirements activity, which is recognized as a critical phase in the software development.

This master's thesis proposes to fulfill the lack of a requirements activity support in Gaia. The proposed solution is the requirements approach: RE4Gaia. This approach is divided into two phases. The *Requirements Model* phase, in which the necessary mechanisms to elicit and model the system requirements are defined. And, the *Requirements Analysis Process* phase, in which the previous specified artifacts are specified with more details. This phase also allows defining the correspondences between the specified requirements and the Gaia analysis artifacts. Our requirements engineering approach for Gaia is developed following the Model Driven Development principles, which allows us to transform in an automatic way a requirements specification into the Gaia analysis models.

The starting point of this work was a Systematic Literature Review. This review had the goal to determine *what* Requirement Engineering Techniques were applied in the existing MAS methodologies and *how* they were applied. The conclusions of this study were used as the basis to build RE4Gaia. The meta-models for RE4Gaia and Gaia were defined, and the corresponding transformations rules between them. In addition, the development process was defined with the goal to guide the developers in the application of this method, including the steps to follow and the artifacts to be generated. Finally, our method was validated by means of a case study.

Agradecimientos

Todos tenemos una vocación en la vida. A mí siempre me ilusionó el llegar algún día a dedicarme a la investigación. A día de hoy tengo que agradecerle a mi director de tesis, Emilio Insfrán, que me sienta más cerca de conseguir esto; gracias a su confianza depositada en mí, permitiéndome entrar en este grupo de investigación, su constante apoyo y todo el entusiasmo transmitido.

A Silvia Abrahão por la colaboración en el desarrollo del Estado del Arte de este trabajo, por las sugerencias en las presentaciones de los congresos y por resolver mis dudas sobre la Evaluación Empírica, y por toda su colaboración y apoyo prestados continuamente.

A Lorena por toda la ayuda prestada de forma desinteresada. Espero que podamos seguir demostrando en el futuro que la Ingeniería del Software tiene muchísimo que aportar a los Sistemas Multi-Agente.

A Isidro Ramos por responder amablemente a mis dudas en su despacho.

A toda la gente del grupo Ingeniería del Software y Sistemas de Información, del Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia. A Adrián, Javier y Sonia por sus sugerencias en las presentaciones y toda la ayuda proporcionada. A mis compañeros de laboratorio: Manolo, Emanuel, Alejandro, Cristóbal, Jose Antonio y Abel.

A mis compañeros de máster. A Rodrigo Lapenda por haberme puesto en contacto con Emilio para realizar esta tesis. A Carlos Cetina por su charla motivadora durante la cena del día antes de mi primera presentación en un congreso. A Sonia Santiago por todos los ánimos previos a los congresos.

Por último, a mi familia, por haber estado en todo momento apoyándome y transmitiéndome ánimo.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

1 Introducción

La historia de la informática ha estado siempre envuelta de cambios continuos. Desde el primer ordenador programable y automático, construido el 12 mayo de 1941 por Konrad Zuse, al primer ordenador personal: el IBM PC, han pasado cinco generaciones de computadores. A partir de la aparición del IBM PC se experimentó una gran expansión de los computadores personales. Gran parte de este cambio se debe a la aparición de los primeros microprocesadores en circuitos integrados en 1971, los cuales permitían reducir drásticamente tanto el tamaño como el costo de estas máquinas permitiendo su adquisición por usuarios domésticos, aumentando además su capacidad y fiabilidad.

Paralelamente, el software incrementó drásticamente su complejidad, para poder adaptarse a la rápida evolución tecnológica del hardware. Este incremento fue progresivo, pasándose de los primeros programas escritos en lenguaje máquina a lenguajes a simbólicos, y de estos a los lenguajes de alto nivel. A estos les sucedieron los lenguajes de programación de cuarta generación, conocidos como los lenguajes no procedurales; en esta generación emergieron con gran impacto los lenguajes de programación orientada a objetos. En la actualidad se está trabajando en lo que se conocerá como quinta generación de lenguajes programación, trabajándose en paradigmas como la programación declarativa, los sistemas expertos o el procesamiento del lenguaje natural.

Dentro de los sistemas futuros de quinta generación, tendrá un protagonismo especial el paradigma de la *Inteligencia Artificial* (IA). La Asociación Americana de Inteligencia Artificial (American Association for Artificial Intelligence), define la Inteligencia Artificial como "*el conocimiento científico de los mecanismos que subyacen al pensamiento y al comportamiento inteligente y su incorporación a las máquinas*". Informalmente podríamos considerar a la IA como una de las áreas de las ciencias computacionales encargadas de la creación de sistemas con comportamiento inteligentes.

Por otra parte, las nuevas exigencias de los sistemas de software actuales han provocado que se demanden cada vez con más frecuencia sistemas interconectados de forma distribuida y concurrente. Aquí juegan un papel importante los *sistemas distribuidos*. Los sistemas distribuidos pueden definirse como una colección de computadores conectados por una red de comunicaciones, que el usuario percibe como un solo sistema .Un ejemplo visible de esta nueva demanda es el gran crecimiento que ha tenido Internet en los últimos años. Hoy en día resulta muy difícil no encontrar un ordenador que no posea la capacidad de conectarse a Internet. Además actualmente han aparecido dispositivos móviles que permiten acceder a Internet desde diferentes tipos de sistemas. Estas nuevas tendencias apuntan a que los sistemas informáticos en el futuro, necesitándose cada vez aplicaciones más complejas.

El contexto tecnológico actual ha incentivado la idea de la creación de sistemas distribuidos con capacidad de ser inteligentes. Un sistema distribuido capaz de reorganizarse podría soportar características como tolerancia fallos o una alta adaptabilidad al entorno. A partir de esta idea surgen los *Sistemas Multi-Agente* (SMA). Los SMA buscan enfrentarse al reto de lidiar con sistemas cada vez más complejos y que requieren frecuentemente de entornos abiertos, elevando el nivel de abstracción sin tener un gran impacto en el coste del producto software. Para dar soluciones a estos retos, los SMA se basan en un modelo donde el control y la información están distribuidos en entidades autónomas capaces interactuar entre sí para conseguir realizar

unos objetivos. Informalmente un SMA puede verse como un sistema distribuido formado por múltiples nodos que en conjunto producen un resultado inteligente. A estos nodos o entidades se les conoce con el nombre de agentes. Los agentes para interactuar exitosamente necesitan: cooperar, coordinar y negociar entrar ellos para realizar sus objetivos.

No existe una definición formal y precisa sobre lo que es exactamente un agente. Algunos autores hablan de agentes inteligentes, definiéndose que son inteligentes si cumplen las siguientes características (Wooldridge M. , An Introduction to Multi-Agent Systems, 2004):

- **Reactividad**. Los agentes inteligentes son capaces de percibir su entorno y responder a cambios que ocurran en él para satisfacer los objetivos con los que fueron diseñados.
- **Pro-actividad**. Los agentes inteligentes son capaces de exhibir comportamiento orientado a sus metas tomando la iniciativa para satisfacer sus objetivos.
- **Sociabilidad**. Los agentes inteligentes son capaces de interactuar con otros agentes para satisfacer sus objetivos.

Algunos autores atribuyen otras características adicionales a un agente. Por ejemplo **Habilidad Social**. Para poder interactuar con su entorno, el agente típicamente deberá interactuar con el mundo exterior. Esta interacción puede darse a distintos niveles, pero en general, el agente necesitará comunicarse con el entorno, con otros agentes, y con sus usuarios (Genesereth & Ketchpel, 1994).

Una de las causas sobre la falta de consenso sobre que es un agente se debe a que el campo de los sistemas Multi-Agente es altamente interdisciplinar. Además de sus influencias provenientes de la IA y los sistemas distribuidos, los SMA tienes influencias de otras áreas como la economía, filosofía, lógica y ciencias sociales. Este hecho causa que existan muchas visiones diferentes sobre los sistemas Multi-Agente. Uno de los Paradigmas relacionados con los SMA es el de la Ingeniería del Software.

El término Ingeniería del Software surgió a finales de 1960 a partir de la Crisis del Software. Esta crisis tuvo origen en la tercera generación del hardware. Fue consecuencia de múltiples factores como: la imprecisión de la estimación de proyectos software y su estimación de costos, la baja calidad y fiabilidad de los productos generados y su difícil capacidad de mantenimiento. La Ingeniería del Software nació con el fin de dar solución a estos problemas, fijándose como objetivos el obtener: una mejora en la calidad de los productos software, aumento en la productividad y una mejora de la capacidad de mantenimiento. Para conseguir estos objetivos, se han propuesto en los últimos años mejoras en las técnicas de modelado, métodos de análisis y diseño, especificaciones formales, desarrollo de herramientas CASE, etc. La IEEE (IEEE Std 610.12-1990, 1993) define la Ingeniería del Software como la "*aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de la ingeniería al software*". Sommerville en una definición más recientes enuncia que: "*la Ingeniería del Software es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de este después que se utiliza*" (Sommerville, 2002).

Las ventajas que ofrecía la Ingeniería del Software no pasaron desapercibidas en la comunidad de desarrollo de SMA. La integración de esta disciplina en el desarrollo de SMA dio origen a lo que se conoce como la *Ingeniería del Software Orientada a Agentes* o *Agent-Oriented Software*

Engineering (AOSE). La investigación en la AOSE se centró en el desarrollo de metodologías para guiar el proceso de desarrollo de sistemas Multi-Agente. La metodologías AOSE, tratan de ofrecer un enfoque disciplinar para analizar, diseñar y desarrollar sistemas Multi-Agente usando métodos y técnicas especificas. Esta aplicación hizo posible que comenzaran a surgir las primeras metodologías para guiar el proceso de desarrollo de sistemas basados en agentes: BDI (Kinny & Georgeff, 1997), Vowel Engineering (Ricordel, 2001), MAS-CommonKADS (Iglesias, Garijo, Centeno-González, & Velasco, 1997). La AOSE ha seguido investigando en esta línea, de modo que en los últimos años han surgido nuevas propuestas metodológicas como Tropos (Castro, Kolp, & Mylopoulos, 2002) y sus posteriores extensiones (Penserini, Perini, Susi, & Mylopoulos, 2007), INGENIAS (Pavón & Gomez, 2003), MaSE (DeLoach, Wood, & Sparkman, 2001), MASSIVE (Lind, 2001) o Gaia (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003). A pesar de las complejidades técnicas que presentan los SMA, la aplicación de técnicas de Ingeniería de Software para guiar el desarrollo de SMA establece un panorama esperanzador para mejorar la calidad, productividad y mantenibilidad de este tipo de sistemas.

1.1 Motivación

La Ingeniería de Requisitos (IR) tiene un impacto crítico en la calidad del Software desarrollado. El éxito de un producto software depende del adecuado establecimiento de las necesidades del usuario y su entorno (Cheng & Atlee, 2007). Un error en la especificación de estas necesidades supondría un incremento en los costes de desarrollo de forma dramática y un retraso en el tiempo de entregar. La IR es un conjunto de actividades concernientes a la identificación y comunicación propósito del sistema, y la determinación del contexto donde será usado.

Por otra parte, Gaia (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003) es una de las metodologías más importantes en el campo de los SMA. Gaia utiliza abstracciones organizaciones definiendo un proceso para el desarrollo de sistemas Multi-Agente. Este proceso comienza con la fase de análisis en la que se pretende especificar la organización computacional. El análisis de la metodología Gaia se centra en transformar los requisitos en un modelo de entorno, modelos preliminares de rol e interacción y un conjunto de reglas organizacionales. A continuación se continúa con la fase de arquitectura, en la que se define la estructura organizacional del sistema. Por último se produce un diseño detallado que tendrá como salida un sistema Multi-Agente listo para implementar en un *framework* de programación de agentes.

A pesar de ser una de las metodologías con mayor impacto en las publicaciones científicas a lo largo de los últimos años, Gaia no da soporte a las actividades de la *Ingeniería de Requisitos*. En Gaia se asume la disponibilidad de los requisitos del sistema y no incluye una fase de adquisición de requisitos. Esto es una desventaja en la integración con otros frameworks de modelado social (Juan, Pearce, & Sterling, 2002). Una vez que se han recopilado los requisitos, Gaia los organiza en forma de estructuras organizacionales, roles y protocolos. Se entra en detalles de bajo nivel como roles y protocolos de forma muy temprana haciendo muy difícil entrando en construcciones muy especializadas muy pronto.

Este proceso podría mejorarse incorporando una fase de modelado de requisitos en la metodología Gaia. El objetivo del modelo de requisitos es entender que debe construirse y

proveer técnicas para capturar adecuadamente las propiedades deseadas para ello (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003).

1.2 Objetivos

En este trabajo, se plantea como objetivo general formular una propuesta para extender la metodología Gaia mediante la construcción de una fase que de soporte al modelado de requisitos previo a la fase de análisis de Gaia. Además, con el objetivo de facilitar la integración de esta nueva fase de requisitos con la metodología Gaia y en consecuencia facilitar la mantenibilidad, se adoptarán técnicas de Desarrollo de Software Dirigido por Modelos, estableciéndose transformaciones automáticas desde los artefactos generados en la fase de requisitos a los artefactos de la fase de análisis de Gaia.

Para satisfacer este objetivo general se plantean los siguientes sub-objetivos:

- Estudiar las soluciones existentes a la adquisición y modelado de requisitos en el campo de los sistemas Multi-Agente.
- Proponer un modelo que provea la expresividad necesaria al usuario para que pueda efectuar la adquisición y modelado de requisitos.
- Aplicar los principios de software dirigidos por modelo, dando soporte a los principios por la arquitectura *Model Driven Architecture* (Object Management Group, 2009). Este sub-objetivo incluirá la definición un meta-modelo de requisitos y un meta-modelo de la metodología Gaia.
- Definir las reglas de transformación de modelos para que puedan transformarse de forma sistemática la especificación de los requisitos del sistema en los modelos de análisis de la metodología Gaia.
- Especificar el Proceso de Desarrollo mediante un lenguaje de modelado de procesos de negocio, para especificar de una forma precisa como utilizar el método propuesto y los artefactos a ser generados.

1.3 Plan de trabajo

En este punto se describirán las sub-tareas en las que se ha dividido este trabajo de tesis con el objetivo de satisfacer los objetivos establecidos en el punto anterior.

Estado del arte

Se estudiaran las metodologías más relevantes dentro del campo de los sistemas compuestos por agentes. Se prestará especial atención a la forma en que resuelven el problema del soporte a las técnicas de requisitos.

Estudio conferencias relevantes

Estudiar a qué conferencias de nivel nacional sería interesante presentar nuestro trabajo. Se estudiaran conferencias dentro del campo de los Sistemas Multi-Agente y la Ingeniería del Software.

Propuesta modelo requisitos

Utilizando la información generada en el estudio del estado del arte, se propondrán los modelos, métodos, técnicas y notaciones necesarios para integrar un mecanismo de modelado de requisitos dentro de la metodología Gaia.

Desarrollo del modelo de proceso

Se definirá el proceso de desarrollo del software propuesto mediante el uso de un lenguaje de modelado de procesos de negocio.

Definición caso de estudio

Se definirá un caso de estudio relevante dentro del campo de los sistemas Multi-Agente. Además se redactará y documentará el enunciado del caso de estudio que se utilizará para probar el modelo propuesto de modelado de requisitos.

Resolución del caso de estudio

Se documentará la aplicación del caso de estudio para el modelo de requisitos propuesto.

Conclusiones del caso de estudio

Se analizarán y se definirán las conclusiones obtenidas a partir de la aplicación del caso de estudio.

Estudio entorno Eclipse

En esta tarea se pretenderá familiarizarse con el entorno de modelado proporcionado por Eclipse dado que será utilizado en las siguientes tareas para definición de los meta-modelos y la definición de las transformaciones.

Definición de los meta-modelos

Se creará un meta-modelo para el modelo de requisitos propuesto.

Definir meta-modelo Gaia

Se creará un meta-modelo de análisis de Gaia.

Definir transformaciones de modelos

Definiremos las transformaciones necesarias para poder obtener de forma automática mediante técnicas de transformación de modelos una especificación Gaia a través de modelo o modelos de adquisición de requisitos propuestos. A continuación de definirán las transformaciones en un lenguaje de transformación de modelos.

Caso de estudio de Gaia completo

Se utilizará el modelo propuesto de adquisición de requisitos en un caso de estudio representativo para razonar sobre los pros y los contras de la propuesta.

Redacción de la memoria de tesis

Se redactará la memoria de tesis documentando todo el trabajo realizado y las conclusiones derivadas de él.

1.4 Planificación

A continuación se comentará la planificación de las tareas que componen este trabajo, tal como muestra la figura 1-1. Adicionalmente se mostrará la duración en días de cada tarea y sus fechas de comienzo y fin.

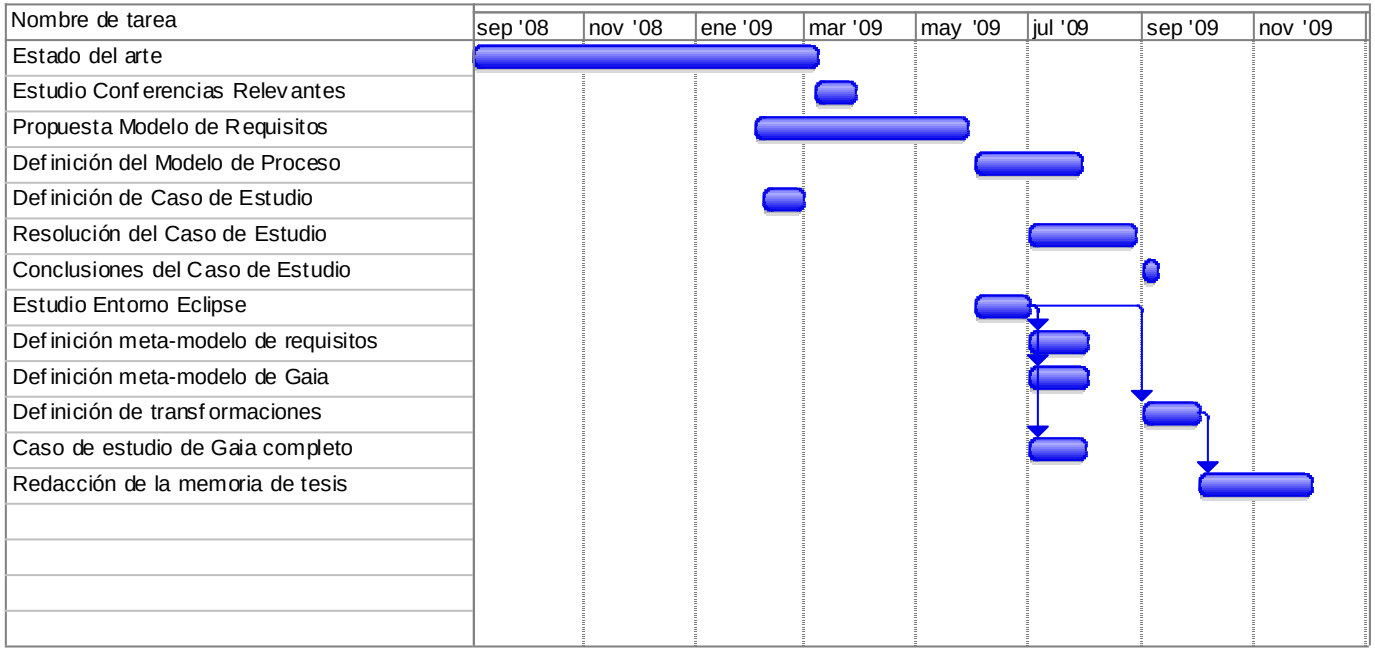


Figura 1-1 Planificación de tareas

1.5 Estructura de la tesis

Esta memoria de la tesis se organiza de la siguiente manera:

- **Capítulo 2**. Fundamentos y Espacios Tecnológicos.

  En este capítulo se hará una introducción a los conceptos necesarios para entender el contexto en el que se desarrolla esta propuesta. Será necesario definir conceptos generales como qué es un sistema Multi-Agente, y concretamente se presentará la metodología sobre la que se basa esta propuesta: la metodología Gaia. Además se incluirán las definiciones dentro de la Ingeniería de Requisitos necesarias para la comprensión de esta propuesta, dado que se trata de un enfoque de modelado de requisitos del sistema. Se incluirán el contexto del Desarrollo de Software Dirigido por Modelos, haciendo énfasis en la arquitectura propuesta por la OMG, la MDA y sus estándares. Por último se hará una introducción a los espacios tecnológicos relacionados con esta tesis: el entorno *Eclipse*, El *Eclipse Modelling Framework* y la herramienta Medini QVT.

- **Capítulo 3.** Estado del Arte.

  En este capítulo se presentará un estado del arte que estará enfocado en el análisis de qué técnicas y notaciones son las más utilizadas para dar soporte a la Ingeniería de Requisitos en las metodologías para desarrollar Sistemas Multi-Agente. Este estado del arte se realizó aplicando una Revisión Sistemática, un tipo de técnica para analizar toda la literatura científica relacionada con un determinado tema de interés con el objetivo de identificar carencias en la investigación actual. Las conclusiones extraídas de este estado del arte se tuvieron en cuenta para el desarrollo de la propuesta.

- **Capítulo 4**. RE4Gaia.

  En este capítulo se presenta RE4Gaia una propuesta para dar soporte a la fase de requisitos. Esta propuesta se divide en dos fases: una fase de *modelado requisitos*, y una fase *de análisis de requisitos*. La información obtenida en esta fase se utilizará para construir los artefactos de la fase de análisis de la metodología Gaia.

- **Capítulo 5**. Desarrollo de Software Dirigido por Modelos y RE4Gaia.

  En el capítulo se presentará la aplicación de técnicas de Desarrollo de Software Dirigido por Modelos en RE4Gaia. En concreto se comentarán los meta-modelos desarrollados respectivamente para R4Gaia y Gaia. Por último se mostrarán las reglas de transformación generadas para dar soporte a las transformaciones automáticas de modelos de modelos RE4Gaia a modelos Gaia.

- **Capítulo 6.** Proceso de Desarrollo**.**

  En este capítulo se describe en la guía metodológica de esta propuesta. En este apartado se incluirá una descripción de los pasos sucesivos actividades y guías para desarrollo de

la especificación de requisitos utilizando RE4Gaia. Para esta definición se utilizará el estándar de la OMG SPEM (Object Management Group, 2005).

- **Capítulo 7**. Caso de Estudio.
  En el capítulo 7 se mostrará la aplicación del caso de estudio para validar la propuesta. El caso de estudio utilizado modela un mercado de pescado, que emplea un proceso de subasta para vender las cajas de pescado.

- **Capítulo 8**. Conclusiones.

  En este capítulo se presentan las aportaciones principales de esta tesis, las publicaciones relacionadas y las líneas de investigación futuras.

- **Anexo A**. Trabajos seleccionados en la Revisión Sistemática .

  En este anexo se recogen las referencias a los artículos analizados en la Revisión Sistemática llevada a cabo con el objetivo de determinar el soporte a la Ingeniería de Requisitos en las metodologías de desarrollo de Sistemas Multi-Agente.

- **Anexo B**. Especificación de los meta-modelos.

  En este anexo se describen los meta-modelos de RE4Gaia y de la fase de análisis de Gaia. Se provee para cada meta-clase una descripción y sus relaciones con otras meta-clases.

- **Anexo C**. Transformaciones de Modelos.

  Este anexo contiene las Transformaciones de Modelos escribas en el Lenguaje *QVT Relations*. Estas transformaciones permiten convertir modelos que conforman el meta-modelo RE4Gaia en modelos de análisis de Gaia.

2 Fundamentos y Espacios Tecnológicos

En este capítulo presentaremos los fundamentos y espacios tecnológicos necesarios para comprender los términos y conceptos de este trabajo.

2.1 Introducción

Esta tesis contiene una propuesta de tratamiento de requisitos para una metodología de desarrollo de sistemas Multi-Agente. Durante este capítulo se explicarán todos los conceptos, estándares y herramientas que estén relacionados directamente con el trabajo planteado a lo largo de la tesis. Concretamente, presentaremos el paradigma de los Sistemas Multi-Agente, el paradigma de la Ingeniería de Requisitos, el Desarrollo de Software Dirigido por Modelos y los espacios tecnológicos relacionados en el trabajo.

El capítulo se estructura de la siguiente forma. En la sección 2.2 se introducirán los Sistemas Multi-Agente y se presentarán las definiciones más importantes relacionadas con este paradigma. En la sección 2.3 se presenta la metodología de desarrollo de Sistemas Multi-Agente Gaia. A continuación en la sección 2.4 se presentará la definición de Ingeniería de Requisitos y sus conceptos clave. La sección 2.5 presentaremos el Desarrollo de Software Dirigido por Modelos, haciéndose especial énfasis en la arquitectura propuesta de la OMG, MDA, y sus estándares relacionados. La sección 2.6 introduce los espacios tecnológicos en los que se apoya esta propuesta. Por último, la sección 2.7 contiene las conclusiones del capítulo. En esta sección se introducirán los conceptos fundamentales y espacios tecnológicos necesarios para entender el contexto que comprende a esta propuesta.

2.2 Sistemas Multi-Agente

Los Sistemas Multi-Agente (SMA) nacen como un nuevo paradigma de la Ingeniería del software para diseñar y desarrollar sistemas de software complejos, adaptándose a las nuevas exigencias de estos. Un SMA es un sistema compuesto por múltiples entidades de software autónomas que interactúan entre sí para conseguir sus objetivos. A estas entidades se les llama Agentes.

Es esta propuesta se empleará la definición de *Agente* propuesta por Wooldridge. Wooldridge (Wooldridge & Jennings, 1994) propone las siguientes características de sistema Multi-Agente:

- *Autonomía:* los agentes operan sin la intervención de humanos, y tienen algún tipo de control sobre sus acciones y estado interno. Normalmente, el agente puede tener un hilo de ejecución interno orientado a realizar una tarea específica. Además el agente tiene la capacidad de decidir qué acción realizar por sí mismo.
- *Habilidad social:* los agentes interactúan con otros agentes y posiblemente humanos a través de algún tipo de lenguaje de comunicación.
- *Reactividad:* los agentes perciben su entorno y responden a los cambios que se producen en él.
- *Pro-actividad:* los agentes no actúan simplemente en respuesta a cambios en su entorno, también son capaces de actuar por iniciativa propia para conseguir sus objetivos.

Estas categorías hacen que los sistemas Multi-Agente adecuados para tratar con las demandas de los sistemas de software complejos emergentes en la actualidad. La *autonomía* refleja la naturaleza descentralizada de los sistemas distribuidos modernos (Tennenhouse, 2000). Además

el comportamiento *reactivo* adaptable al entorno y *pro-activo* para conseguir los objetivos para los que fue diseñado, hacen que los agentes sean adecuados para operar en los entornos impredecibles donde se espera que operen (Zambonelli F. , Jennings, Omicini, & Wooldridge, 2001).

Otro concepto que es preciso comprender es el de *sistema abierto*. En este tipo de sistemas los agentes no están diseñados para compartir un objetivo común y puede que hayan diseñados por gente con diferentes objetivos. Además la composición del sistema puede variar dinámicamente y los agentes pueden entrar y salir del sistema. En oposición al concepto de sistema abierto, en un *sistema cerrado* no se permite a ningún agente externo entrar a participar en la organización. En estos sistemas se conocen a priori los agentes que van a participar en el sistema.

Por último la Ingeniería del Software Orientada a Agentes, conocida por sus siglas en inglés *Agent Oriented Software Engineering (AOSE)*, es el área de la ingeniería del software orientada al estudio de sistemas con agentes. Dentro de esta área se han propuesto múltiples metodologías para guiar el proceso de desarrollo del software como: Tropos (Penserini, Perini, Susi, & Mylopoulos, 2007), INGENIAS (Pavón & Gomez, 2003), MaSE (DeLoach, Wood, & Sparkman, 2001), MASSIVE (Lind, 2001), ANEMONA (Giret, Botti, & Valero, 2005) y GORMAS (Argente, Botti, & Julián, DCAI 2009). En la sección 2.3 introduciremos Gaia (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003), una de las metodologías más relevantes dentro de la AOSE, la cual está relacionada directamente con nuestra propuesta.

2.3 La Metodología Gaia

La primera versión de que Gaia fue propuesta por Wooldridge *et al.* en (Wooldridge, Jennings, & Kinny, The Gaia Methodology for Agent-Oriented Analysis and Design, 2000). Esta primera versión se creó con el objetivo de dar soporte al análisis y diseño de Sistemas Multi-Agente, quedando no soportadas las fases de especificación de requisitos e implementación. En esta versión, un sistema Multi-Agente es visto como una organización computacional de agentes que interactúan entre sí para alcanzar un objetivo común de la aplicación.

El **análisis** se compone de los modelos de rol e interacción. En el modelo del roles, para cada rol se especifica una plantilla compuesta de sus actividades, protocolos, permisos y responsabilidades. Las actividades son funciones que puede llevar a cabo un agente por sí mismo. En oposición a una actividad, un *protocolo* precisa de la interacción con uno o más roles. Los *permisos* especifican de qué forma puede acceder un rol un a un *recurso*, estableciendo límites en su ámbito ejecución. Las *responsabilidades* expresan su nacionalidad y condiciones invariantes que deben cumplir en todo momento. En la fase de **diseño** se crean los modelos de agente, servicio y comunicación. El *modelo de agente*, se agrupan los distintos roles en tipos de agente. El *modelo de servicio*, está compuesto por múltiples servicios que son bloques de funcionalidad derivados de las actividades los protocolos. Por último en el modelo comunicación, se definen las relaciones entre los diferentes agentes.

Esta versión fue extendida por Zambonelli *et al.* en (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003), extendiendo la propuesta original con conceptos organizacionales. En este documento cuando hagamos referencia a Gaia, asumiremos que se trata de la extensión oficial de Zambonelli y es sobre la que basaremos nuestra propuesta.

La metodología Gaia explota las abstracciones organizacionales, definiendo un proceso de diseño de Sistemas Multi-Agente. El proceso de Gaia comienza en la fase de análisis, cuyo objetivo es crear una especificación que será la base del diseño de la organización computacional. El proceso continua con el diseño de la arquitectura, la cual aspira a definir la estructura del sistema organizacional en términos de topología y régimen de control. Por último se lleva a cabo el diseño detallado que producirá una especificación de un sistema Multi-Agente independiente de tecnología. Esta especificación puede implementarse fácilmente empleando un *framework* de programación de agentes.

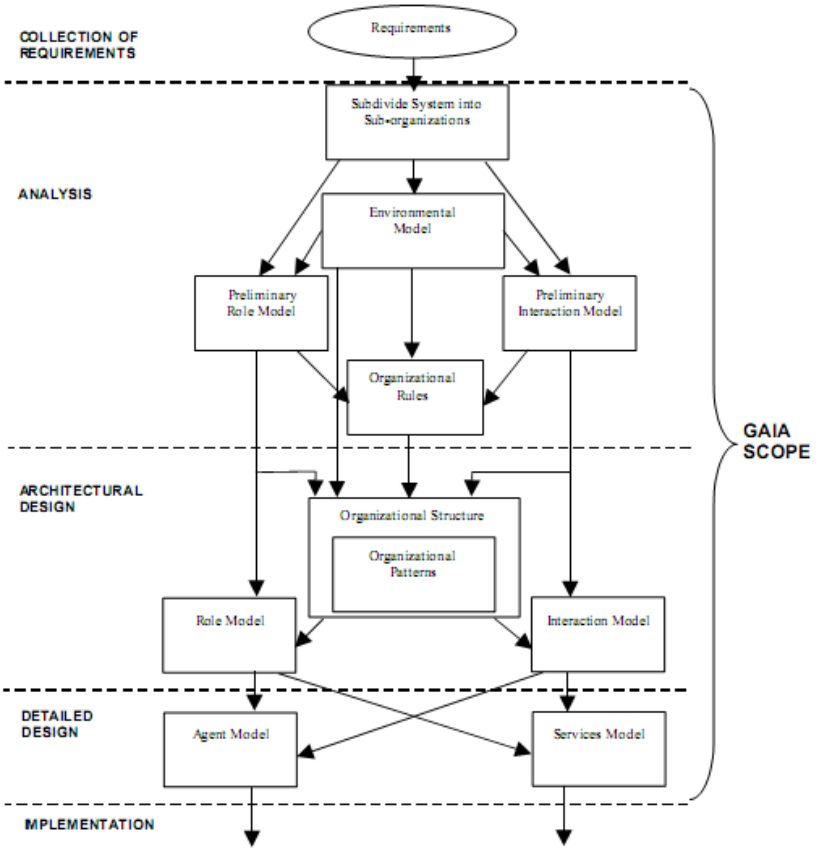


Figura 2-1 Modelos y relaciones entre ellos en Gaia[1]

2.3.1 Fase de Análisis

La fase de análisis pretende recopilar todas las especificaciones y requisitos para construir el sistema Multi-Agente. Para llevar a cabo esto se pretende identificar las posibles sub-organizaciones que compondrán el sistema global y para cada una de ellas producir cuatro modelos abstractos básicos:

- El **modelo de entorno**, el cual captura las características del entorno operacional de un sistema Multi-Agente.
- Un **modelo preliminar de roles** para capturar las actividades que ejecutaran en el MAS.

[1] Figura extraída de (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003)

- Un **modelo preliminar de interacciones** preliminar para capturar interdependencias básicas entre roles.
- Un conjunto de **reglas organizacionales** expresando restricciones y directivas que restringirán al sistema Multi-Agente globalmente.

2.3.1.1 Las Organizaciones

El primer paso en el análisis de Gaia atañe determinar si tienen que coexistir múltiples organizaciones. Podemos encontrar una sub-organización si existen porciones del sistema global que:

- Exhiben un comportamiento específicamente orientado a un sub-objetivo.
- Interactúan débilmente con otras partes del sistema.
- Requieren competencias que no se necesitan en otras partes del sistema.

2.3.1.2 Modelo Preliminar de Roles

A pesar de que no se conoce la estructura de la organización en la fase de análisis, es posible identificar algunas características del sistema independientes de la organización. Estas características derivarán en los modelos preliminares de roles e interacciones.

Gaia representa los roles con una descripción abstracta y semi-formal el comportamiento esperado de una entidad. Un rol se representa mediante dos tipos de atributos: permisos y responsabilidades.

Los protocolos se escribirán con subrayado para diferenciarlos de las actividades.

Permisos

Los objetivos principales de los permisos son:

- Identificar los recursos a los que puede acceder el rol legítimamente.
- Establecer los límites en los que el rol puede operar.

Los permisos relacionan los roles con el entorno en el que están situados. De forma adicional pueden representar información o un conocimiento que puede adquirir un agente.

Para representar un permiso, Gaia usa la misma representación utilizada para representar recursos del modelo de entorno.

Las Responsabilidades

Estos atributos pretenden determinar el comportamiento esperado de un rol y los atributos claves asociados con un rol.

Se dividen en dos tipos:

- **Responsabilidades de viveza**. Intuitivamente afirman que “algo bueno pasa”, es decir describen aquellos estados que debe proporcionar un agente dadas ciertas condiciones.
- **Responsabilidades de seguridad**. Aseguran que “nada malo pasa”, es decir que nos mantenemos en estados aceptables. Un formalismo muy utilizado para especificar este

tipo de propiedades es la lógica temporal. No obstante Gaia utiliza simplemente predicados para especificar propiedades de seguridad.

Si bien el formalismo más ampliamente utilizado para especificar las responsabilidades es la lógica temporal, (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003) propone, por razones de utilidad industrial, una alternativa basada en la utilización de expresiones regulares. Mediante éstas, las propiedades de vida se especifican mediante expresiones de vida o vivacidad, que definen el ciclo de vida del rol.

Donde los elementos atómicos son las actividades y los protocolos. Estos a su vez se conectan mediante los conectores mostrados en la Tabla 2-1, formándose así las expresiones de viveza.

Estas expresiones tienen la forma general

Nombre de rol = expresión

Tabla 2-1 Operadores para expresiones de viveza

Operador	Interpretación
x.y	X seguido por y
X | y	Ocurrirán x o y
X*	X ocurre cero o más veces
X+	X ocurre una o más veces
x^{ω}	X ocurre infinitamente
[x]	X es opcional
X || y	X e y intercaladas

Plantillas de Roles

Gaia propone la siguiente plantilla para representar los roles. Dado que en esta fase es un modelo preliminar, no es necesario completar todos los campos si el desarrollador lo considera necesario.

Tabla 2-2 Plantilla de Roles en Gaia

Rol: *nombre del rol*
Descripción: *descripción textual del rol y su comportamiento*
Protocolos y actividades: *lista de protocolos de actividades que ejecuta*
Permisos: *Lee: recursos sobre los que puede leer* *Modifica: recursos sobre los que puede modificar* *Consumir: recursos que puede consumir*
Responsabilidades:
Viveza: Lista de Responsabilidades de Viveza
Seguridad: Lista de Responsabilidades de Seguridad

2.3.1.3 Modelo preliminar de interacciones

El modelo de interacción de Gaia captura las dependencias y relaciones entre varios roles de una organización Multi-Agente. Debido a que el modelo de roles en esta fase es preliminar, el modelo de interacción también será preliminar.

En Gaia un protocolo puede ser visto como un patrón de interacción institucionalizado. Una definición de protocolo consta de los siguientes atributos:

- **Nombre del protocolo:** descripción textual que captura la naturaleza de la interacción.
- **Iniciador:** el rol o roles responsables de empezar la interacción.
- **Participante:** el rol o roles que interactúan con el iniciador del protocolo.
- **Entradas:** información usada por el rol iniciador mientras lleva a cabo el protocolo.
- **Salidas:** información proporcionada por el rol que responde durante la interacción.
- **Descripción:** descripción textual explicando el propósito del protocolo y las actividades procesadas implicadas en su ejecución.

Al igual que en el modelo de roles, podemos recopilar las propiedades de un protocolo en la siguiente plantilla:

Tabla 2-3 Plantilla de Protocolo

Nombre del Protocolo: *descripción breve de la interacción*		
Iniciador: *rol o roles responsables de iniciar la interacción*	**Participante:** *rol o roles interactúan con el iniciador*	**Entradas:** *información de entrada del protocolo*
Descripción: *descripción textual que explica el propósito del protocolo*		**Salidas:** *información d salida del protocolo*

2.3.1.4 Reglas organizacionales

Los modelos preliminares de roles y protocolos capturan los patrones de interacción, funcionalidades y características básicas independientemente de la estructura organizacional. Sin embargo existen relaciones entre roles, protocolos o roles y protocolos que se capturan de forma más precisa con reglas organizacionales.

Gaia considera que las reglas organizacionales son responsabilidades de la organización. El concepto de responsabilidad es similar al empleado en la definición de las responsabilidades de un rol. En consecuencias las reglas organizacionales se dividirán en dos tipos: reglas de viveza y reglas de seguridad.

- Las **reglas de viveza** definen como la dinámica de la organización debería evolucionar a lo largo del tiempo. Estas reglas podrán hacer referencia a otras expresiones de viveza pertenecientes a roles diferentes.

- Las **reglas de seguridad** definen invariantes que debe respetar la organización independientemente del tiempo.

Debido a la similitud, las reglas organizacionales pueden expresarse usando el mismo formalismo adoptado para especificar reglas de seguridad y vivacidad en los roles.

2.3.2 Fase de Diseño

La salida de la fase de análisis de Gaia documenta todas las características funcionales que ha de expresar el sistema Multi-Agente, junto a las características del entorno operacional en el cual el sistema Multi-Agente se ha de situar. Esta especificación estructurada se utiliza en el diseño de la arquitectura para identificar una estructura de la organización del sistema Multi-Agente.

Escogiendo estructura organizacional

La elección de la estructura organizacional es una fase crítica en el desarrollo de sistemas Multi-Agente. Para escoger adecuadamente las características de la organización se han de seguir una serie de principios:

- Eficiencia del sistema: es la necesidad de que un sistema Multi-Agente trate la computación y complejidad de la coordinación de un problema.
- La necesidad de respetar las reglas organizacionales.
- La necesidad de minimizar el salto a la organización en el mundo real.

Cualesquiera que sean los factores específicos que pueden determinar la elección de una estructura organizacional, es altamente probable que las mismas hayan sido ya consideradas en el pasado, en algún contexto general, y que hayan conducido a consideraciones y elecciones similares. En este caso, se vuelve particularmente efectivo el contar con catálogos de "estructuras organizacionales", posiblemente modulares.

Representando la estructura organizacional

Una vez hemos identificado una estructura organizacional del sistema Multi-Agente que se adapte a nuestro problema, hemos de determinar cómo representarla.

Para especificar una organización formalmente hemos de explicar las relaciones entre roles que existen en él y sus tipos.

A pesar de que no existe una ontología aceptada para relaciones organizacionales, podemos identificar ciertos tipos de relaciones que suelen ocurrir en las organizaciones Multi-Agente:

- ***Control***. Identifica una relación de autoridad entre roles.
- ***Peer***. Identifica una relación entre agentes del mismo estatus.
- ***Depends_on***. Identifica las formas en que un agente puede depender de otro.

Es posible incluir una representación gráfica para complementar las relaciones.

Perfeccionamiento del modelo de roles e interacciones

Una vez se ha definido la estructura organizacional, ya es posible completar los modelos preliminares de roles e interacciones identificados en la fase de análisis. La finalidad de estos modelos es:

- Definir todas las actividades en las que un rol participa, considerando sus responsabilidades de viveza y seguridad.
- Definir aquellos roles que presencia no se identificó en la fase de análisis y que cuya identificación se derive de la estructura organizacional.

- Completar la definición de protocolos requerida por las aplicaciones, especificando en que roles se ve envuelto el protocolo.
- Definir aquellos protocolos organizaciones que se derivan de la estructura organizacional.

La única diferencia entre las representaciones preliminares y las completas será que ahora se incluirán todas las características de los roles y los protocolos.

Definición del modelo de agentes

Un agente es un entidad de software activa que juega un conjunto de roles de agente. A partir de esto, definiremos el modelo de agentes como el modelo que pretende identificar que clases de agentes se definen para jugar los roles específico y cuantas instancias de cada clase se tienen que instanciar en el sistema actual.

En general es posible que existen relaciones entre roles y agentes de uno a uno. Sin embargo un diseñador puede escoger empaquetar varios roles en el mismo tipo de agente si lo considera necesario por razones de eficiencia y simplicidad en el diseño. Es importante tener en cuenta que no se contempla la herencia en el modelo de agentes de Gaia.

De forma adicional el modelo de agentes documenta las instancias que aparecerán en el sistema anotando el número de instancias que aparecerán para cada clase de tipo de agente. Para esto se pueden utilizar los calificadores de instancia propuestos:

Tabla 2-4 Calificadores de instancia

Calificador	Significado
n	Tendrá exactamente *n* instancias
m..n	Tendrá entre *m* y *n* instancias
*****	Cero o más instancias
+	Una o más instancias

El modelo de servicios

El propósito del modelo de servicios es identificar los servicios asociados con cada clase de agente.

En orientación a objetos, un servicio correspondería a un método, pero en orientación a agentes un servicio no está disponible para ser invocado por otros agentes en la forma en que habitualmente un método lo está en una clase de objetos. Debido a la naturaleza intrínsecamente activa de los agentes, se entiende a los servicios como un bloque único y coherente de actividades en la cual el agente se ve involucrado. Tal servicio no tiene que ser necesariamente activado por un evento externo (pues la autonomía propia de los agentes implica que existen capacidades de decisión interna).

Para cada servicio es necesario documentar sus propiedades. Debemos identificar las entradas, salidas, pre-condiciones y post-condiciones. Las entradas y salidas se derivan del modelo de protocolos. De forma análoga las pre y post condiciones se derivan respectivamente de las propiedades de seguridad de un rol.

Los servicios que componen un agente se derivan de la lista de protocolos, actividades, responsabilidades y propiedades de vida de los roles que implementa.

GAIA no prescribe una implementación para el modelo de servicios, sino que lo deja liberado al desarrollador, el cual, por ejemplo, puede implementar cada servicio directamente como un método en un lenguaje orientado a objetos, o bien puede descomponer un servicio en un conjunto de métodos.

2.4 Ingeniería de Requisitos

2.4.1 Definiciones

El éxito de un sistema software depende del éxito del entendimiento de las necesidades del usuario y su entorno. Para comprender estas necesidades, la *Ingeniería de Requisitos* (IR) juega un papel clave en el desarrollo del software. El éxito en la aplicación de técnicas de Ingeniería de Requisitos influye directamente en la calidad del producto software desarrollado. Según Zave (Zave, 1997), "*La IR es la rama de la ingeniería del software que trata con el establecimiento de los objetivos, funciones y restricciones de los sistemas software. Asimismo, se ocupa de la relación entre estos factores con el objeto de establecer especificaciones precisas.*"

Para entender qué es la IR, es necesario definir el concepto de requisito. Existen múltiples definiciones sobre lo que es un *requisito de software*. De acuerdo con el estándar IEEE 610.12-1990 (IEEE Std. 610.12-1990, 1990), un requisito puede ser:

a) Una condición o capacidad necesaria para un usuario para resolver un problema o conseguir un objetivo.

b) Una condición o capacidad que debe reunir o poseer un sistema o componente de un sistema para satisfacer un contrato, estándar, especificación, u otro documento formalmente impuesto.

c) Una representación documentada de una condición o capacidad como las definidas en a) o b).

Los requisitos pueden ser:

- **Funcionales**, indican características y restricciones sobre la funcionalidad del software.
- **No funcional**es, definen restricciones sobre los requisitos no funcionales como fiabilidad, seguridad, usabilidad, etc.

Por otra parte, una *especificación de requisitos del software* (SRS) es una descripción completa del comportamiento del sistema a desarrollar. La SRS debería especificar qué funciones son realizadas sobre qué datos para producir qué resultados y para quién. Sin embargo una SRS no entrar en aspectos de diseño como: particionamiento del software en módulos o elección de estructuras de datos.

2.4.2 Actividades de la IR

De acuerdo con los el trabajo Betty Cheng y Joanne Atlee (Cheng & Atlee, 2007), la IR puede dividirse en varias actividades principales: adquisición de requisitos, modelado, análisis de requisitos, validación y verificación, y gestión de requisitos. A continuación se comentarán las actividades más importantes en relación a este trabajo, incluyendo las principales notaciones y técnicas relacionadas con cada una de ellas.

2.4.2.1 Adquisición

La *adquisición* de requisitos suele ser la primera en la fase del proceso de IR. Esta actividad abarca las actividades que permiten comprender las metas, objetivos y motivaciones para construir el sistema software propuesto. Esto incluye el establecimiento de los límites del sistema. Los límites definen a un alto nivel, los límites del entorno operacional del sistema. Existen múltiples técnicas y notaciones propuestas para obtener esta información:

- *Identificación de stakeholders*. Dentro de la actividad de adquisición de los requisitos del sistema, puede incluirse la identificación de los *stakeholders*. El término *stakeholder*, hace referencia a quienes pueden afectar o son afectados por las actividades de una empresa. Un *stakeholder* puede ser un cliente, desarrollador o un usuario.
- *Metas*. Denota los objetivos que un sistema debe alcanzar. Identificar las metas de alto nivel tempranamente en el sistema es una tarea crucial en el desarrollo de software. La adquisición orientada a metas, es una actividad donde los requisitos de alto nivel como metas de negocio, se refinan en metas de más bajo nivel como metas técnicas que eventualmente se operacionalizan en un sistema.
- Los *escenarios* son otro tipo de notación, que consisten en descripciones parciales y concretas del comportamiento de un sistema en una determinada situación.
- Los *requisitos no funcionales*, son una notación. Este tipo de requisitos son también conocidos como requisitos de calidad dado que son un conjunto de características de calidad que se han de tener en cuenta durante el desarrollo de un sistema software.
- La *metáfora* nos permite estructurar conceptos a partir de otros.

2.4.2.2 Modelado

El *modelado* permite expresar una especificación de requisitos en términos de uno o más modelos. Estos modelos tienden a ser más precisos, completos y claros que los modelos realizados durante la adquisición de requisitos. El proceso de creación de modelos precisos ayuda a identificar detalles que no se identificaron en la actividad de adquisición de requisitos.

Las *notaciones* de modelado ayudan a establecer límites en los niveles de abstracción en las descripciones de requisitos, proveyendo un vocabulario y reglas estructurales que encajan con las entidades, relaciones comportamiento y restricciones del problema que está siendo modelado. Entre las notaciones de modelado encontramos:

- *Modelos de Objeto*. Proponen una descripción de un conjunto de estados, normalmente finitos, donde cada estado representa una configuración de objetos.
- *Modelos de comportamiento* proveen descripciones abstractas del comportamiento esperado del sistema.

2.4.2.3 Análisis de Requisitos

En estas técnicas se incluyen las actividades para evaluar la calidad de la especificación de requisitos. Algunas analizan errores de mal formación de la especificación, donde se entiendes como "errores" la ambigüedad, inconsistencia o no completitud. Otras técnicas analizan anomalías como interacciones desconocidas entre requisitos, obstáculos posibles para la satisfacción de los requisitos y razonamientos perdidos.

En esta actividad se incluyen técnicas como (Cheng & Atlee, 2007): *checklists, análisis de consistencia, análisis de conflictos, gestión de riegos o análisis de variabilidad*.

2.4.2.4 Validación y Verificación de Requisitos

La validación de requisitos se encarga de que los modelos y la documentación expresen correctamente las necesidades de los *stakeholders*. La validación suele hacerse típicamente de manera subjetiva, debido a la documentación de los requisitos de manera informal. Este tipo de validaciones requieren que el usuario tenga que participar activamente en el proceso de validación, revisando directamente los artefactos.

Para los casos en que la especificación de los requisitos se haya hecho de forma formal, se pueden aplicar técnicas de verificación para comprobar que esta satisface las necesidades de los stakeholders. En las técnicas de verificación se pueden considerar:

- *Model checking.* Se comprueba el comportamiento del modelo contra alguna propiedad de lógica temporal en las trazas de ejecución.
- *Model satisfiability.* Se comprueba que existen instancias validad de modelos de objetos restringidos, y que las operaciones de los modelos de objetos preservan invariantes.

2.4.2.5 Gestión de Requisitos

La gestión de requisitos es una actividad que incluye varias técnicas para gestionar los requisitos, incluyendo la evolución de los requisitos a través del tiempo. Un tema de especial interés, es el de las herramientas y técnicas que dan soporte parcial a la tarea de identificar y documentar las relaciones de trazabilidad entre los artefactos de requisitos y los artefactos de diseño. En la actividad de gestión de requisitos también se incluyen las técnicas para determinar la madurez y estabilidad de los requisitos adquiridos, para que los requisitos que se posible que cambien se puedan aislar.

2.5 Desarrollo de Software Dirigido por Modelos

2.5.1 Introducción

El **Desarrollo de Software Dirigido por Modelos** (DSDM) es una aproximación de desarrollo de sistemas de software basada en la creación de modelos. Un modelo permite definir la funcionalidad, estructura y/o comportamiento de un sistema. Los modelos permiten trabajar a un nivel de abstracción más cercano a los conceptos del dominio, en lugar de centrarse en conceptos orientados a plataforma como ocurre con el desarrollo de software tradicional. El objetivo de esta propuesta es maximizar la productividad, maximizando la interoperabilidad entre sistemas, facilitando la reusabilidad y la adaptación del sistema a cambios tecnológicos.

2.5.2 La Arquitectura Dirigida por Modelos (MDA)

En este trabajo se aplicará la propuesta de DSDM propuesta por el Object Management Group (Object Management Group, 2009), la **Arquitectura dirigida por Modelos** o *Model Driven*

Architecture (MDA). MDA se basa en cuatro principios fundamentales (Beydeda, Book, & Gruhn, 2005):

1. Los modelos se expresan es una notación bien definida que es la clave del entendimiento del sistema para soluciones a nivel de organización.
2. La construcción de los sistemas se puede organizar entorno a un conjunto de modelos imponiendo una serie de transformaciones entre ellos, organizada en un marco de trabajo de capas y transformaciones.
3. Se da soporte para describir modelos en un conjunto de meta-modelos que facilita la integración y transformación entre modelos, y es la base de la automatización mediante herramientas.
4. La adopción y aceptación de este enfoque basado en modelos precisa de estándares industriales para proveer accesibilidad a los clientes y fomentar la competitividad entre proveedores.

Sobre los estándares comentados en el punto 4, en MDA se recomienda el uso de estándares propuestos por la OMG: MOF, UML, CWM, QVT, etc.

2.5.2.1 Modelos en MDA

MDA propone la creación de tres tipos de modelos:

- **Platform Independent Model (PIM)**. Un modelo de un sistema que es independiente de la información acerca de la plataforma o tecnología que es usada para implementarlo.
- **Platform Specific Model (PSM)**. Un modelo de un sistema que incluye información acerca de la tecnología específica que se usará para su implementación sobre una plataforma especifica.
- **Code Model (CM)**. Dado que el PSM es dependiente de plataforma, se puede automatizar la generación de código.

2.5.3 Meta-Object Facility (MOF)

MOF es un estándar propuesto por la (Object Management Group, 2009) para dar soporte a la arquitectura MDA. En este estándar se propone una arquitectura de meta-modelado basado en cuatro capas. Si consideramos un modelo como una abstracción de un fenómeno en el mundo real; el meta-modelo es una abstracción donde se reflejan las propiedades del modelo. El meta-modelado se utiliza básicamente en la utilización de modelos para definir otros modelos.

Respecto a la arquitectura de MOF, se distinguirán las siguientes capas:

- **Nivel M3: Meta-metamodelado**. En esta capa reside el meta-metamodelo, un lenguaje usado para definir los meta-modelos del nivel M2. Dentro de OMG, MOF es el lenguaje estándar utilizado en esta capa.
- **Nivel M2**: **Meta-modelo**. Los meta-modelos de la capa M2 se utilizan para describir los modelos del nivel M1. Por ejemplo el meta-modelo de UML (Object Management Group, 2009) describiría al propio UML.
- **Nivel M1**: **Modelo**. En este nivel se definen los modelos. Un modelo es una instancia de un meta-modelo del nivel M2. Por ejemplo, si en el M2 residiera el meta-modelo de UML en el nivel M1 podríamos tener uno de sus modelos: Modelo de Clases, Diagrama de Actividad, etc.
- **Nivel M0**: **Instancias**. En esta capa se describen objetos del mundo real.

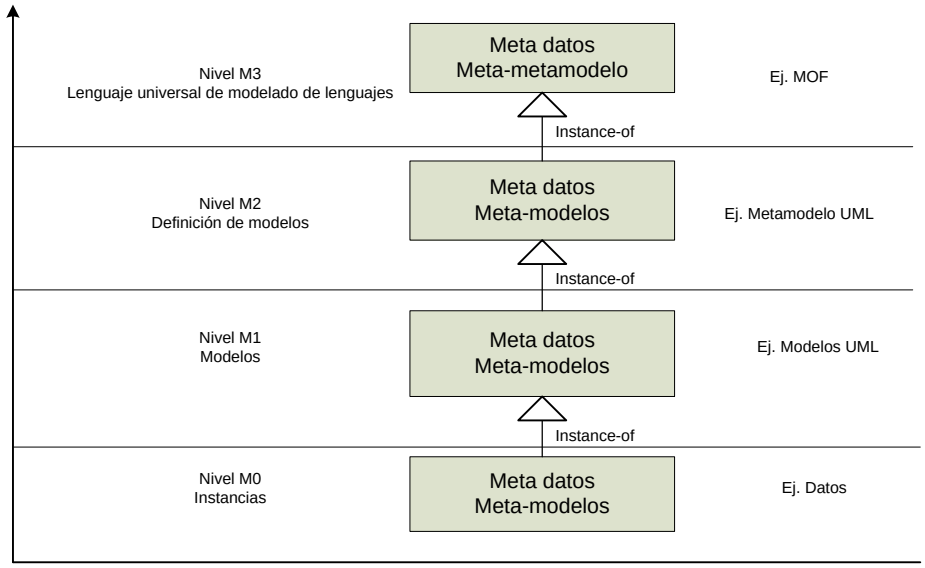


Figura 2-2 Niveles en la arquitectura MOF

MOF se considera una arquitectura de meta-modelado cerrada. Esto quiere decir que el último nivel, el M3, se pueden definir con instancias de elementos de M3. Esto implica que MOF se puede definir a sí mismo.

Además MOF provee los siguientes conceptos para definir un lenguaje:

- **Clases**, las cuales modelan los meta-objetos MOF.
- **Tipos de Datos**, modelan la necesidad de datos descriptivos.
- **Asociaciones**, modelan relaciones binarias entre meta-objetos.
- **Paquetes**, modularizan los objetos.

La OMG ha definido dos variantes de MOF:

- *EMOF for Essential MOF*. Subconjunto de MOF, con el objetivo de proveer un framework para mapear modelos MOF a implementaciones como JMI o XMI para meta-modelos simples.
- *CMOF for Complete MOF*. El modelo CMOF se utiliza para definir meta-modelos como UML2. Está construido a partir de EMOF y el Core:Constructs de UML2.

2.5.4 Object Constraint Language

El ***Object Constraint Language*** **(OCL)** es un lenguaje formal que puede utilizarse para definir restricciones o consultas sobre modelos en términos de lógica de predicados. OCL fue desarrollado por IBM como lenguaje de modelado de negocio. Surgió a partir del método Syntropy, un lenguaje de modelado de negocios de la *IBM Insurance Division*, desarrollado por Steve Cook y John Daniels.

Usualmente se suele utilizar para definir expresiones sobre modelos UML. Estas expresiones suelen especificar condiciones invariantes que deben cumplirse en el sistema que se está modelando o definir consultas sobre los objetos descritos en un modelo. Cuando se evalúa una expresión OCL, no se producen efectos secundarios (es decir, su evaluación no puede alterar el estado del sistema de ejecución correspondiente).

OCL no solo se puede aplicar para modelos UML, también es posible aplicarlo a meta-modelos UML o MOF, ya que están expresados en UML. Aquí el OCL puede usarse para restringir la semántica del meta-modelo, por ejemplo basándose en estereotipos.

En el contexto MDA, OCL puede utilizarse de tres maneras posibles:

- Construir modelos más precisos, en el nivel M1 de MOF.
- Definición de Lenguajes de Modelado.
- Especificar Transformaciones.

2.5.5 QVT (Query/View/Transformation)

Query/Viev/Transformation **(QVT)** es un estándar para transformación de modelos definido por la OMG (Object Management Group, 2009). Una transformación de modelos es el proceso de convertir un modelo origen, conforme a un meta-modelo origen, en un modelo destino, conforme a un meta-modelo destino. Las transformaciones de modelos están basadas en reglas. Estas reglas enlazan construcciones del modelo origen a construcciones en el modelo destino y se definen a nivel de meta-modelo. Cuando el meta-modelo origen y el destino sean los mismos diremos que es una transformación endógena. Si son meta-modelos distintos, diremos que es una transformación exógena.

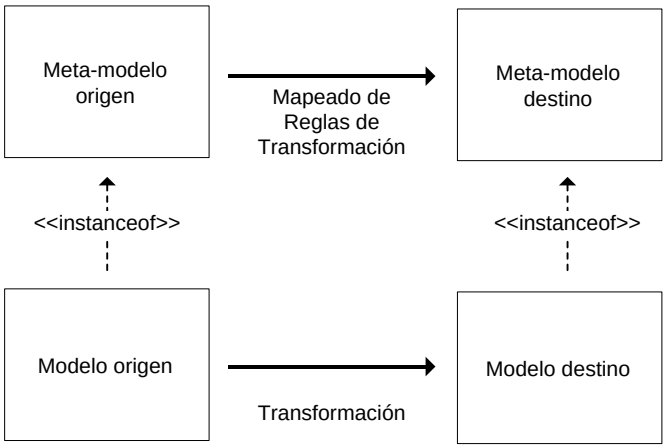


Figura 2-3 Esquema Transformación

Arquitectura QVT

La especificación QVT de la OMG (Object Management Group, 2008) tiene una naturaleza hibrida imperativa / declarativa. La parte declarativa se divide en una arquitectura de dos niveles. Esta parte declarativa provee el *framework* para semántica operacional de la parte imperativa. La figura 2-4 muestra la relación entre los meta-modelos que componen esta arquitectura.

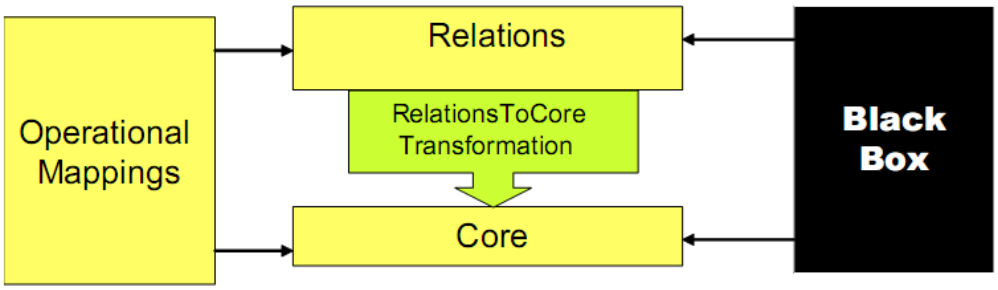


Figura 2-4 Relaciones entre los meta-modelos QVT

Arquitectura Declarativa de Dos Niveles

La arquitectura declarativa se divide en dos capas:

- El meta-modelo ***Relations*** y un lenguaje que soporta *pattern-matching* y creación de plantillas. Las trazas entre los elementos del modelo implicados en las transformaciones, se crean implícitamente.
- Un meta-modelo ***Core*** y un lenguaje definido utilizando extensiones mínimas de EMOF y OCL. Todas las clases de trazas se definen explícitamente como modelos MOF y la creación y destrucción de instancias de trazas se definen de la misma manera que la creación y destrucción de un objeto.

La semántica de Relations se define en combinación de inglés y lógica de predicados de primer orden. *Core* solo soporta *pattern matching* sobre un conjunto de variables, evaluación condiciones en estas variables contra un conjunto de modelos. Es posible implementar el modelo Core directamente o usarlo como referencia para la semántica de *Relations*, las cuales mapearían a Core utilizándose a sí mismo como lenguaje de transformación.

Arquitectura Imperativa

Este lenguaje permite provee dos mecanismos para soportar implementaciones imperativas utilizando *Relations* o *Core*: un lenguaje estándar –***Operational*** *Mappings*- o implementaciones no estándares conocidas como *implementaciones operacionales MOF de caja negra* o *implementaciones* ***Black Box***.

El lenguaje *Operational* provee una forma estándar de proveer implementaciones imperativas. Puede utilizarse cuando existen una o más relaciones que son difíciles de definir utilizando una especificación puramente declarativa. Por su parte, es posible derivar implementaciones de *Relation*s como implementaciones *Black-Box* para "conectar" cualquier implementación con la misma signatura.

Lenguaje QVT Relations

En el lenguaje *Relations*, las transformaciones entre los modelos candidatos se especifican como un conjunto de relaciones que deben cumplirse para que la transformación tenga éxito. Un modelo de candidato es un modelo que conforma un tipo de modelo, el cual es una especificación de qué tipo de elementos del modelo puede tener para conformar a ese modelo, de manera similar a un tipo variable que especifica qué tipo de valores puede conformar una variable en un programa.

Una **relación** declara restricciones que deben satisfacerse por los elementos de los modelos candidatos. Una relación define dos o más dominios y un par de predicador *when* y *where*, que especifican una relación que debe cumplirse entre los modelos candidatos.

Un **dominio** es un tipo de variable que puede coincidir con un modelo de un tipo de modelo dado. Un dominio tiene un patrón, que puede verse como un grafo de nodos objeto, cuyas propiedades y relaciones de asociación tienen como origen la instancia de un tipo de dominio.

El siguiente ejemplo extraído de la especificación de QVT (Object Management Group, 2008) muestra como dos dominios se corresponden con los elementos de los modelos "uml" y "rdbms":

```
relation PackageToSchema   /* map each package to a schema */
  {
    domain uml p:Package {name=pn}
    domain rdbms s:Schema {name=pn}
  }
```

Una clausula ***when:*** especifica una condición que debe cumplir una relación. En contraste, una clausula ***where*** específica una condición que debe satisfacerse por todos los elementos del modelo que participan en la relación. Este tipo de clausulas pueden contener restricciones en el lenguaje OCL. En el ejemplo, la relación *ClassToTable* solo se aplicará cuando se satisfaga la relación *PackageToSchema* entre el paquete que contiene la clase y el esquema que contiene la tabla. Además, la clausula *where* indica que todos los elementos que participan la relación deben de satisfacer la relación *AttributeToColumn*.

```
relation ClassToTable   /* map each persistent class to a tab
  {
    domain uml c:Class {
        namespace = p:Package {},
        kind='Persistent',
        name=cn
     }
    domain rdbms t:Table {
         schema = s:Schema {},
          name=cn,
          column = cl:Column {
              name=cn+'_tid',
              type='NUMBER'},
          primaryKey = k:PrimaryKey {
```

```
            name=cn+'_pk',
            column=cl}
    }
    when {
      PackageToSchema(p, s);
    }
    where {
      AttributeToColumn(c, t);
    }
  }
```

Por último las clausulas *checkonly* y *enforce* determinan si se debe forzar el aplicar una relación en el dominio destino. Con ***checkonly*** se comprueba si existe una correspondencia valida en el modelo que satisfaga la relación. Con ***enforce*** si la comprobación falla, el modelo destino se modifica para que se satisfaga la relación.

2.6 Espacios Tecnológicos

En este punto se presentarán los espacios tecnológicos relacionados con el trabajo de tesis.

2.6.1 Eclipse

La comunidad Eclipse es una comunidad de código abierto, cuyos proyectos se centran en la creación de una plataforma de desarrollo abierta formada por *frameworks* extensibles, herramientas para crear, desplegar y gestionar software a lo largo del ciclo de vida.

Uno de sus proyectos más relevantes es Eclipse. Eclipse es un proyecto de código abierto, robusto, con múltiples características, con calidad comercial y una plataforma industrial para el desarrollo de herramientas altamente integradas. El *Entorno Desarrollo Integrado* (IDE) utiliza diferentes módulos para enriquecer su funcionalidad, esto supone una ventaja frente a otro tipo de entornos monolíticos donde la funcionalidad no es configurable. Por ejemplo añadiendo el *plug-in* Subversion se podía disfrutar de forma adicional de un sistema de control de versiones integrado en el entorno desarrollo.

En definitiva, la naturaleza esta herramienta la convierte en un IDE abierto y extensible para múltiples propósitos. En este trabajo tendrá especial interés el *Eclipse Modelling Framework* (EMF), un *framework* que permite gestionar modelos y generación de código a partir de modelos descritos en XMI. EMF se describe detalladamente en el punto 2.6.2.

2.6.2 EMF

El proyecto EMF es un *framework* de modelado y facilidades para generación de código para construir herramientas y otro tipo aplicaciones basadas en modelos de datos estructurados. Se parte de una especificación de un modelo descrito en XMI. EMF provee herramientas para producir un conjunto de clases de Java para el modelo. Además se generan un conjunto de clases adaptadoras que permiten vistas y ediciones basadas en comandos del modelo.

Los modelos pueden especificarse utilizando Java anotado, documentos XML, o herramientas de modelado como *Rational Rose* gracias a que se puede importar a partir de ellas a EMF. Lo más importante de todo, es que *EMF* provee las bases para establecer interoperabilidad con otras herramientas y aplicaciones basadas en *EMF*.

En relación con la *Object Management Group* (OMG) y si *Meta Object Facility* (MOF), EMF tiene relación con ellos. De hecho EMF empezó como una implementación de la especificación MOF madurada a partir de la experiencia obtenida en el desarrollo de herramientas por parte de los desarrolladores de Eclipse. *EMF* puede ser visto como una implementación eficiente en Java de uso conjunto de la API de *MOF*. Sin embargo para evitar confusiones, el Meta-modelo basado en el núcleo *MOF* de *EMF* se llamará *Ecore* (The Eclipse Foundation, 2009) . En la figura 2-5 se muestra una vista de los componentes del *Ecore*.

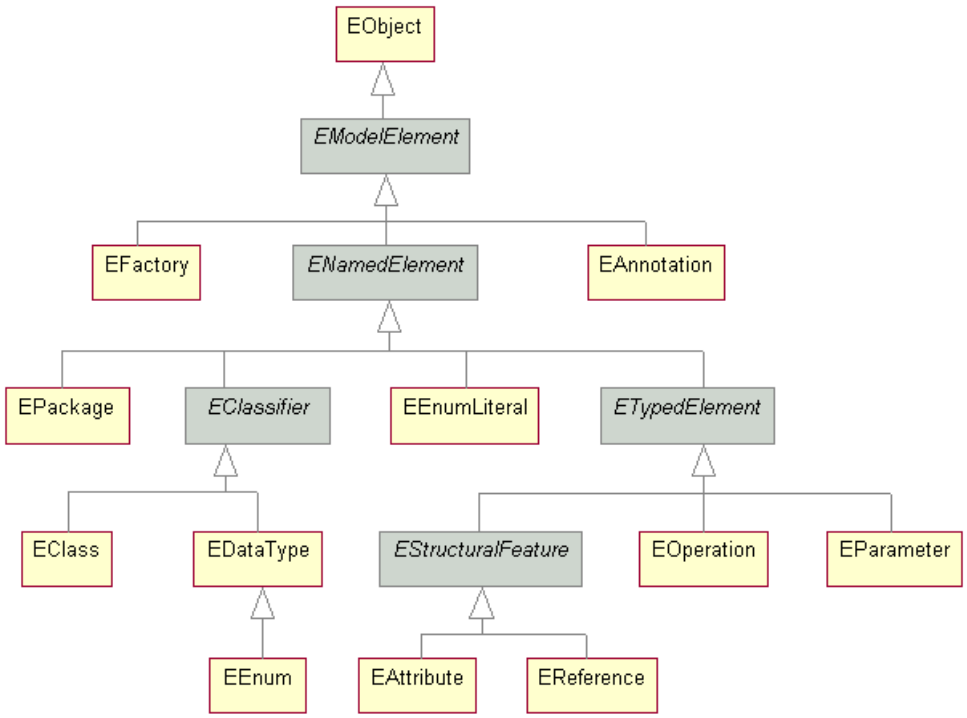


Figura 2-5 Componentes del Ecore[2]

En la propuesta actual de MOF 2.0 el *Essential MOF* (EMOF), un subconjunto similar del modelo MOF, se ha separado. Existen pequeñas diferencias, principalmente los nombres, entre Ecore y EMOF; sin embargo, EMF puede leer y escribir señalizaciones de EMOF.

2.6.3 Medini QVT

En este punto se utilizará la herramienta utilizada durante este trabajo de tesis: *Medini* QVT. *Medini QVT* es un conjunto herramientas que soporta transformaciones de Modelo modelo. *Medini* implementa el estándar de la OMG, QVT Relations. Este estándar está diseñado para especificar transformaciones de modelo a modelo para permitir un desarrollo ágil, mantenimiento y representa personalización del proceso especificación de reglas de transformación.

---

[2] Figura extraída de (The Eclipse Foundation, 2009)

Algunas de sus características más importantes son:

- La ejecución de las transformaciones QVT se muestran como una sintaxis concreta textualmente del lenguaje *Relations*.
- Posee un editor con asistente de código.
- Soporta el depurado paso a paso de *Relations*.
- La gestión de trazabilidad permite actualizaciones incrementales durante la re-transformación.
- Transformaciones bidireccionales.

2.7 Conclusiones

En este capítulo se han introducido los conceptos clave para entender el marco conceptual que comprende esta tesis. Se han presentado los principales paradigmas que comprenden a este trabajo y los espacios tecnológicos los que se utilizaron para darle soporte.

Los sistemas Multi-Agente nacen con la idea de dar soporte a problemas complejos y distribuidos que serían difíciles de resolver en otros paradigmas. Un sistema multi agente está compuesto por múltiples entidades interacción entre sí para conseguir un objetivo. A estas entidades la llamaremos agentes, las cuales según Wooldridge (Wooldridge & Jennings, 1994) deberían tener las siguientes características: autonomía, habilidad social, reactividad y pro-actividad. La Ingeniería del Software Orientada a Agentes (AOSE), es la rama de la ingeniería del software centrado en el estudio sistemas compuestos por agentes. Dentro de la AOSE se han propuesto muchas metodologías para guiar proceso de desarrollo de sistemas Multi-Agente. Entre estas metodologías se encuentra Gaia, una de las más importantes dentro del paradigma.

Gaia (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003) se basa en la metáfora organizacional y se basa en la visión del sistema como una organización computacional. Gaia soporta las fases de análisis y diseño en el desarrollo del software, quedando fuera de la misma las fases de especificación de requisitos e implementación. La fase de análisis incluye la división del sistema en su-organizaciones, la creación del modelo entorno, la construcción del Modelo preliminar de roles y la creación del Modelo preliminar e interacciones. Por último se especifican las reglas de la organización. La fase de diseño se especifica la estructura organizacional, se refinan los modelos de roles interacciones, y se definen el modelo de agentes y de servicios.

Por otra parte, de acuerdo con Dave (Zave, 1997): "*La IR es la rama de la ingeniería del software que trata con el establecimiento de los objetivos, funciones y restricciones de los sistemas software. Asimismo, se ocupa de la relación entre estos factores con el objeto de establecer especificaciones precisas".* La ingeniería requisitos incluye las actividades de: adquisición, modelado, análisis, validación y verificación y gestión de requisitos.

El desarrollo seguro dirigido por modelos es una aproximación desarrollo de software basadas en el uso modelos de forma que se trabaja a un nivel de abstracción mayor. Esto permite mejorar aspectos como la interoperabilidad, reusabilidad, mejorando la productividad y adaptación a cambios tecnológicos. Esta propuesta adoptará la propuesta de DSDM propuesta por la OMG, la arquitectura MDA. MDA se basa en la definición de modelos bien definidos mediante técnicas de meta-modelado y transformaciones automáticas de modelos mediante

reglas de transformaciones. Para ello recomienda el uso de un conjunto de estándares propuesto: MOF, UML, CWM, QVT, etc.

Respecto al espacio tecnológico, esta propuesta se basará en la propuesta de desarrollo Eclipse. En concreto se utilizará el *Eclipse Modelling Framework* para utilizar técnicas de meta-modelado y definición de transformaciones automáticas de código. La herramienta utilizada será *Medini QVT*, debido a que entre otras ventajas posee soporte a transformaciones bidireccionales y un motor de transformaciones basado en el lenguaje de transformaciones *QVT Relations*.

3 Estado del Arte

En este capítulo se analizarán los trabajos relacionados que proponen metodologías para construir sistemas Multi-Agente, haciendo especial énfasis en como dan soporte a las técnicas de Ingeniería de Requisitos.

3.1 Introducción

La Ingeniería de Requisitos juega un papel clave en el desarrollo del software cuando pretendemos identificar las necesidades del cliente. Los enfoques tradicionales para la identificación, modelado y análisis de los requisitos del usuario para ciertos tipos de sistemas, pueden no ser totalmente efectivos. Un sistema Multi-Agente es un tipo específico de sistema compuesto por múltiples agentes que interactúan entre sí. Estos sistemas pueden utilizarse para resolver problemas que serían difíciles o incluso imposibles de abordar en sistemas de un agente individual o sistemas monolíticos. La Ingeniería del Software Orientada a Agentes o *Agent Oriented Software Engineering* (AOSE) ha surgido como un nuevo paradigma de desarrollo de software y se han propuesta nuevas metodologías para guiar el proceso de desarrollo de software. Estas metodologías están influenciadas por campos diferentes como la Psicología, Biologías y la Ciencia Social (Bernon, Cossentino, & Pavón, 2005).

Esta variabilidad hace necesaria la clasificación de estas metodologías de acuerdo a las técnicas empleadas. Aunque existen algunas publicaciones científicas que comparan metodologías de desarrollo de SMA, existe una carencia de estudios específicos orientados en métodos, técnicas y notaciones de la Ingeniería de Requisitos en el desarrollo de SMA.

Previamente se realizaron algunas revisiones para intentar resolver este problema. Sin embargo en el campo de la AOSE, muchos de los estudios propuestos son revisiones de literatura informales con criterios de comparación subjetivos, algunos de los cuales solo incluyen un pequeño número de metodologías y técnicas, conceptos o notaciones de Ingeniería de Requisitos aplicadas en el desarrollo de los SMA.

Dentro de estas revisiones, Hederson-Sellers y Gorton (Henderson–Sellers & Gorton, 2002), se describe una genealogía de algunas de las metodologías de AOSE actuales. Este trabajo clasifica las metodologías en:

i. Independientes de las metodologías Orientadas a Objetos.
ii. Extensión de metodologías Orientadas a Objetos, con el objetivo de dar soporte a los conceptos relacionados con los agentes.

De una manera similar, Sudeikat *et al.* (Sudeikat, Braubach, Pokahr, & Lamersdorf, 2004) describen una genealogía con tres posibles ancestros para las metodologías AOSE: Orientadas a Objetos, Ingeniería del Conocimiento e Ingeniería de Requisitos. Este trabajo propone un framework de comparación con cuatro grupos para realizar la selección: conceptos, notaciones, procesos y pragmatismo. En (Silva, Tedesco, Castro, & Pinto, 2004), Silva *et al.* proponen un enfoque nuevo para comparar metodologías para SMA usando un *framework* de Requisitos No Funcionales. En (Cernuzzi, Cossentino, & Zambonelli, 2005) hace una clasificación de algunas metodologías para SMA de acuerdo al ciclo de vida adoptado.

En nuestro trabajo se opto por realizar una Revisión Sistemática de la literatura con el objetivo de evitar los problemas de las comparaciones realizadas previamente. Una de las grandes ventajas de una Revisión Sistemática es que es un método objetivo y repetible para la determinación de actividades de investigación recientes. Una **Revisión Sistemática** (RS) de Literatura significa la identificación, evaluación e interpretación de todas las investigaciones disponibles que son relevantes para una pregunta de investigación particular, área temática o fenómeno de interés (Kitchenham, 2004). Las Revisiones Sistemáticas se han aplicado exitosamente en muchos campos de la Ingeniería del Software (por ejemplo: Ingeniería de Requisitos (Davis, Dieste, Hickey, Juristo Juzgado, & Moreno, 2006), Ingeniería Web (Mendes, 2005)).

En (Blanes, Insfrán, & Abrahão, International Symposium on Distributed Computing and Artificial Intelligence, 2009), se describieron brevemente los resultados del primer estudio en este tema, aplicando una Revisión Sistemática que consideraba tres criterios para analizar los trabajos de investigación publicados en dos librerías digitales (ACM DL y IEEExplore). Posteriormente en (Blanes, Insfrán, & Abrahão, Requirements Engineering in the Development of Multi-Agent Systems: A Systematic Review, 2009) se realizó una revisión utilizando cuatro librerías digitales ACM Digital Library (ACM), IEEExplore Electronic Database (IEEExplore), Inspec (IE), y Science Direct (SD) analizando seis criterios y re-utilizando el trabajo hecho en la primera revisión. En esta tesis se ha incluido todo el proceso de revisión para que se pueda comprender el contexto en el que tienen validez los resultados y las causas que originaron llegar a estas conclusiones. Además dado que la revisión sistemática es un método repetible, futuros trabajos relacionados con este tema pueden utilizar como punto de partida este estudio.

En esta sección se comentarán los resultados de la segunda revisión sistemática realizada con el objetivo de determinar la actividad de investigación actual en la Ingeniería de Requisitos aplicada al desarrollo de SMA, tomando como fuente primaria cuatro bibliotecas digitales (ACM DL, IEEExplore, Inspect y Science Direct). Esta sección se estructura de la siguiente manera. La sección 3.2 describe el método usado en este trabajo para analizar los trabajos existentes sobre los SMA y la Ingeniería de Requisitos y presenta los resultados obtenidos de la Revisión Sistemática. Finalmente, la sección 3.3 presenta las conclusiones de este capítulo.

3.2 Revisión Sistemática

3.2.1 Método de Investigación

Con el objetivo de proveer un enfoque más objetivo para realizar revisiones en la literatura científica, seguiremos en enfoque presentado en (Kitchenham, 2004) para realizar Revisiones Sistemáticas de Literatura. Las actividades incluidas en una Revisión Sistemática pueden agruparse en tres fases: *planificación, ejecución y comunicación de los resultados*. En la *planificación*, se identifican las razones para realizar la revisión, se definen las cuestiones de investigación y se definen los protocolos de revisión. En la fase de *ejecución*, se seleccionan los estudios primarios, se definen las estimaciones de calidad y ejecutan la extracción y supervisión de datos, por último los datos obtenidos se sintetizan. Finalmente, en la fase de *presentación de resultados*, se especifican los mecanismos de difusión y se presenta el reporte de revisión. Las actividades que atañen a la planificación y ejecución se describen respectivamente en las sub-secciones de la 3.2.2 a la3.2.6, el informe de los resultados se presenta en la sección 3.2.7.

3.2.2 Preguntas de Investigación

El objetivo de nuestra revisión es estudiar los trabajos actuales de desarrollo de SMA dando respuesta a la siguiente Pregunta de Investigación: *"¿Qué técnicas de Ingeniería de Requisitos se aplican para dar soporte a la actividad de Ingeniería de Requisitos en el desarrollo de SMA y cómo se aplican?"*.

Esta pregunta de investigación nos permitirá resumir el conocimiento actual sobre el uso de técnicas de Ingeniería de Requisitos en el desarrollo de SMA e identificar carencias en la investigación actual, permitiendo sugerir nuevas áreas de investigación futura o proponer la integración de estas técnicas en el análisis y diseño de metodologías de desarrollo de SMA.

Con el objetivo de entender la Pregunta de investigación, en (Kitchenham, 2004) se recomienda considerar los siguientes cuatro puntos de vista:

i. **Población:** artículos de investigación presentando metodologías para desarrollar sistemas Multi-Agente.
ii. **Intervención**: métodos y técnicas de Ingeniería de Requisitos.
iii. **Resultados**: no se pretende conseguir ningún resultado en concreto.
iv. **Diseño experimental**: ninguno.

Esta revisión es más limitada que una RS completa por el hecho de que no se incluirán los artículos referenciados en los artículos incluidos y no se incluirán fuentes de literatura gris.

3.2.3 Identificación y Selección de Estudios Primarios

En este trabajo se han utilizado cuatro librerías digitales científicas como fuentes de estudios primarios: ACM Digital Library (ACM), IEEExplore Electronic Database (IEEExplore), Inspec (IE), y Science Direct (SD). Además se ha incluido en las fuentes un libro de metodologías de AOSE (Henderson-Sellers & Giorgini, 2005), publicado en el año 2005. Desafortunadamente algunas publicaciones como las actas del congreso de la *Advancement of Artificial Intelligence* (AAAI) o trabajos publicados en Springer, debido al hecho de que no teníamos acceso a ellas. Entendemos que ese hecho podría implicar una reducción en la validez y propósito de este estudio.

Respecto a la cadena de búsqueda, se probaron con diferentes cadenas hasta que se obtuvo una que recuperaba el mayor número de artículos relevantes. La cadena de búsqueda fue:

((multiagent or multi-agent or "multi agent" or "agent-based") and (methodology or method or process or approach) and ("requirements elicitation" or "requirements modeling" or "requirements modeling" or "requirements analysis" or "requirements specification")). Esta cadena se usó en ACM, IEEExplore, IE, y SD. Esta cadena incluyó revistas profesionales y actas de congresos desde 1998 a marzo de 2009.

3.2.4 Criterios de Inclusión y Procedimientos

Los criterios de inclusión pretenden identificar aquellos estudios primarios que proveen evidencias directas sobre la pregunta de investigación. Se han incluido artículos con propuestas de técnicas, métodos y notaciones para tratar con la actividad de Ingeniería de Requisitos durante el desarrollo de SMA. Se excluyeron los artículos que presentaban:

i. Desarrollo de herramientas específicas.
ii. El desarrollo de plataformas de agentes.

iii. Evaluación y comparación de metodologías AOSE.
iv. Métodos para el desarrollo de arquitecturas.
v. *Short papers*.
vi. Artículos introductorios para sesiones especiales, *workshops* o conferencias.

3.2.5 Estrategia de Extracción de Datos

Los datos extraídos se han comparado de acuerdo a la pregunta de investigación enunciada anteriormente, la cual se descompone en los criterios presentados en la Tabla 3-1.

Tabla 3-1 Pregunta de Investigación y Criterios Correspondientes

Pregunta	Opciones
¿El trabajo propuso un sistema nuevo/adaptado de IR?	Método Nuevo, Método Adaptado.
¿A qué actividades de la IR se da soporte?	Adquisición, Modelado y Análisis.
¿Qué conceptos y notaciones se emplean?	Metas, Casos de Uso o Escenarios, RNF, Modelos de Objetos, Modelos ER, Modelos de Comportamiento, Métodos Formales, Otros.
Técnicas Empleadas	Identificación de *Stakeholders*, Metáforas, Patrones, Ontologías.
Soporte a la trazabilidad	Pre-requisitos, Post-requisitos.
Soporte Automatizado	Especificación, Soporte a la Transformación a Artefactos de Diseño. Validación, *Baseline*.
Validación de la propuesta	Caso de estudio, Estudios empíricos, Ninguno.

Respecto al primer criterio, su objetivo es determinar si el artículo propone un **método de Ingeniería de Requisitos** *nuevo* o *adaptado*. Muchas metodologías de desarrollo de SMA han utilizado técnicas y conceptos de otros paradigmas como el orientado a objetos o la ingeniería de requisitos adaptándolos para el paradigma de los SMA; otras sin embargo han propuesto métodos completamente nuevos para tratar con las técnicas de IR.

El segundo criterio concierne a las **tareas soportadas de la IR**: adquisición, modelado y análisis. Esta clasificación está basada en la utilizada en los trabajos de (Nuseibeh & Easterbrook, 2000) y (Cheng & Atlee, 2007). La *adquisición* comprende a todas las actividades que permiten entender las metas, objetivos y funcionalidades de alto nivel necesarias para construir el sistema de software propuesto. El *modelado* permite expresar los requisitos en términos de uno o más modelos. Finalmente en el *análisis* se evalúa la calidad de la especificación de requisitos y se valida que se adapte correctamente a los requisitos del usuario. Se excluirán de este criterio: la fase de v*alidación y verificación*, al considerarse que esta fase está principalmente relacionada con el campo de las descripciones formales y no es relevante para este trabajo, y la fase de *gestión de requisitos*, dónde sólo consideraremos relevante para el trabajo las técnicas relacionadas con la trazabilidad, las cuales ya se analizarán en detalle en el quinto criterio.

En el criterio tercero se analizan los **conceptos y notaciones** usadas para identificar y modelar los requisitos para el sistema software a construir. Algunos trabajos utilizan metas, escenarios o

requisitos no funcionales como frameworks conceptuales para identificar los requisitos del usuario. Los *modelos de objetos, de entidad-relación o de comportamiento* también son alternativas para la identificación y modelado de requisitos. Dentro de la categoría de modelos de comportamiento se encuentran los diagramas de actividad, *workflows*, diagramas de estado y modelos de tareas.

El cuarto criterio analiza las **técnicas empleadas** para tratar con los requisitos, incluyendo *identificación de stakeholders, metáforas, patrones y ontologías*. El quinto criterio, la **trazabilidad**, definida como la "habilidad para seguir la vida de un requisito en ambos sentidos, hacia sus orígenes o hacia su implementación, a través de las especificaciones generadas durante el proceso de desarrollo" (IEEE Std 830-1998, 1998). Se categorizará el soporte a la trazabilidad de acuerdo a la clasificación propuesta por Gotel y Finkelstein (Gotel & Finkelstein, 1994), distinguiendo entre *pre-trazabilidad* y *post-trazabilidad*.

El sexto criterio es el **soporte automatizado** a las técnicas de requisitos. Aquí se distinguirá entre facilidades de automatización para la *especificación, transformación a artefactos* de diseño, soporte a la *validación* (ejemplo: corrección, consistencia), definición de *baseline*.

El último criterio analiza cómo se han **validado** los trabajos propuestos. Consideraremos: cados de estudio (estudios que eximan un fenómeno o unidad, recopilando datos y analizando los resultados para un solo caso) o estudios empíricos (pruebas de la hipótesis contra un experimento donde no hay influencia del observador.

3.2.6 Ejecución de la revisión

La revisión de llevo a cabo extrayendo como estudios primarios resultados de ACM, IEEExplore, IE y SD. Esta revisión se dividió en dos rondas. La primera se realizó el 20 de diciembre de 2008, y la segunda se ejecutó el 6 de marzo de 2009 (añadiendo los nuevos resultados a la lista). La aplicación del protocolo de revisión produjo 58 documentos (la lista completa puede encontrarse en el Anexo A).

La búsqueda en base de datos bibliográficos produjo 835 publicaciones potenciales (300 de ACM, 12 de IEEE, 83 de IE y 440 de SD). Después de aplicar el criterio de exclusión (sección 3.2.4), se seleccionaron 48 publicaciones: 30 de AMC, 8 de IEExplore, 2 de IE y 8 de SD.

La diferencia significante entre el número de documentos extraídos de ACM y IEEEXplore se debe posiblemente al hecho de que ACM incluida muchas de las conferencias relacionadas con los SMA como: AAMAS, IAT, ICSE, etc. El bajo número de publicaciones seleccionadas de IE y SD se debe al hecho que muchos de los resultados ya ha aparecían en ACM e IEEE. Adicionalmente, se encontraron 13 publicaciones relevantes del libro (Henderson-Sellers & Giorgini, 2005). Después de aplicar el criterio de exclusión, se seleccionaron 10 artículos de la búsqueda manual.

3.2.7 Resultados

En esta sección se realizará el informe de los trabajos observados tras la aplicación de la revisión sistemática. La Tabla 3-2 presenta los resultados obtenidos de este estudio. Se han agrupado por criterio de selección y fuente de publicación.

La figura 3-2 muestra el número de artículos clasificados por fuente y año. El diagrama de barras muestra que existe un interés creciente en la ingeniería de requisitos desde el año 2001 (aunque se encontró un artículo de 1998).

Entre los trabajos revisados, un 79% utilizaba *métodos adaptados* de otros paradigmas y un 21% proponía *nuevos* métodos para dar **soporte a los requisitos** del sistema.

Tabla 3-2 Resultados de la Revisión Sistemática

Criterio de Selección		ACM	IEEE	EI	SD	Libro	Total
Soporte a la IR	Nuevo	9	0	0	2	1	12
	Adaptado	21	8	2	6	9	46
Fase en la que se da soporte a la IR	Adquisición	15	3	1	6	1	26
	Modelado	21	4	2	6	10	43
	Análisis	29	7	2	7	10	55
Notaciones y Conceptos Empleados	Metas	23	4	1	6	6	40
	Escenarios	8	3	2	2	7	22
	RNF	11	4	0	3	1	19
	Modelos de Objetos	6	1	2	4	5	18
	Entidad-Relación	0	0	0	1	0	1
	Modelos de Comportamiento	7	2	1	3	5	18
	Métodos Formales	10	2	1	1	0	14
	Otros	11	3	2	2	4	22
Técnicas Empleadas	Identificar *Stakeholders*	10	0	0	2	0	12
	Metáforas	1	1	0	0	1	3
	Patrones	11	1	0	6	5	23
	Ontologías	4	0	0	2	2	8
Trazabilidad	Pre-requisitos	0	1	0	1	0	2
	Post-requisitos	9	2	2	6	1	20
Soporte Automatizado	Especificación	9	2	1	3	7	22
	Validación	8	1	0	2	3	14
	Transformación hacia Artefactos de Diseño	4	1	1	1	3	10
	Definición de *Base Line*	0	0	0	0	0	0
Validación	Caso de Estudio	18	4	1	5	9	37
	Estudio Empírico	3	0	0	0	0	3
	Ninguno	9	4	1	3	1	18

ACM - ACM Digital library
IEEE - IEEEXplore electronic database
EI - Inspect

SD - Science Direct
Libro – Libro (Henderson-Sellers & Giorgini, 2005)

Entre las **actividades soportadas de la IR**, la más soportada fue para el análisis (95%), seguido por el modelado (74%), y la adquisición (45%). Este resultado indica que muchas de las metodologías de desarrollo de SMA prestan más atención a las tareas de modelado y análisis (ver figura 3-1a). Respecto a la tarea de *adquisición*, 17 de los 26 trabajos estuvieron basados o integrabas alguna parte de i*. Este es el caso de *GORMAS* (Argente, Botti, & Julián, DCAI 2009) que integra el análisis de requisitos de Tropos, extendiéndolo con una categorización de

temas (misión del sistema, metas funcionales y metas operativas). Un enfoque diferente se presenta en (Burmeister, Arnold, Copaciu, & Rimassa, 2008), donde se aplican conceptos de agentes a nivel de Procesos de Negocio. Otra alternativa al *framework* i* es el trabajo de Ranjan y Misra (Ranjan & Misra, 2006), donde se ofrece adquisición de requisitos usando cuatro modelos: modelo de producto, modelo organizacional, modelos de usuarios del sistema y el modelo cognitivo. Otra alternativa es *HOMER* (Wilmann & Sterling, 2005), una técnica de adquisición de requisitos para metodologías de AOSE. Finalmente, el trabajo de Fuentes-Fernandez et al (Fuentes-Fernández, Gómez-Sanz, & Pavón) propone la *Requirements Elicitation Guide*. Esta guía propone una serie de preguntas que los clientes y los desarrolladores deben responder para identificar y definir los requisitos de una SMA.

Respecto a las **notaciones y conceptos** (ver figura 3-1b), la más utilizada fue la basada en metas (69%). Otras alternativas fueron: Escenarios (38%), RNF (33%), Modelos de Comportamiento (31%), Modelos de Objetos (31%) y Métodos Formales (24%). En este contexto hemos de destacas el uso intensivo de Modelos de Roles detectados en las metodologías SMA (Dehlinger & Lutz, 2005), (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003), (Ranjan & Misra, 2006) y el hecho que sólo uno de los artículos revisados (Wagner, 2003) empleo el modelo Entidad-Relación.

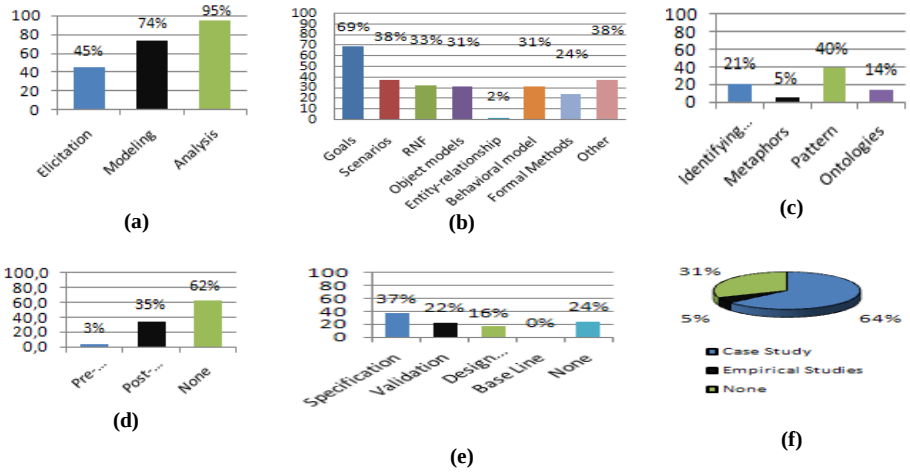


Figura 3-1 Resultados de la Revisión Sistemática

Respecto al soporte de la **trazabilidad** (ver figura 3-1d), el 62% de los artículos no daban ningún soporte explicito a la trazabilidad. La *post-trazabilidad* solo se aplicó en un 35% de los casos, mayoritariamente en trasformaciones de requisitos a artefactos de diseño o código fuente. Solamente el trabajo de Paraense et al (Paraense, Gudwin, & Gonçalves, 2007) aplicó trazabilidad, concretamente desde un *workflow* operacional (a nivel de proceso de negocio) había un diagrama de flujo cognitivo.

Respecto al **soporte automático** a la IR (ver figura 3-1e), el 38% de los artículos mencionaban herramientas para dar soporte automatizado a la *especificación*. Adicionalmente, un 24% de los artículos proveían algún tipo de *validación* automática de la especificación. Sin embargo el 21& de los artículos no proveen *ningún* tipo de soporte automático. Además, un 17% de los artículos *soportan transformaciones de requisitos a artefactos de diseño*. No se ha encontrado ninguna herramienta que soporte la definición de *baseline*.

Respecto al método de evaluación de las propuestas (ver figura 3-1f), el caso de estudio ha sido el método de validación más utilizado (64%). Algunos trabajos no ofrecían ningún tipo de validación (31%). Los trabajos que solo ofrecían ejemplos de su enfoque se incluyeron en esa categoría. Finalmente, solo un 5% de los trabajos validaron su trabajo con estudios empíricos (experimentos controlados).

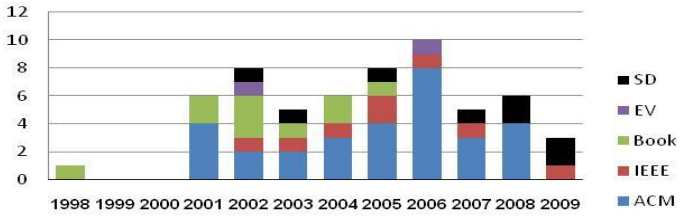


Figura 3-2 Número de publicaciones por año y fuente

3.3 Conclusiones

En esta sección se han presentado los resultados de la revisión sistemática con el motivo de investigar qué técnicas y métodos de IR para el desarrollo de SMA se han empleado en los últimos 10 años. La revisión se ha llevado a cabo como una revisión sistemática porque es un método objetivo y repetible de evaluación de publicaciones científicas.

Los resultados han identificado varias carencias en la investigación. Específicamente, la mayoría de las metodologías de SMA se enfocan en el modelado y análisis de requisitos y no en su adquisición. Para la adquisición de requisitos, el método i* ha sido el enfoque mayormente utilizado. Este hecho revela que hay una carencia de métodos y técnicas alternativas para adquirir los requisitos apropiadamente en los SMA.

Respecto a los conceptos y notaciones, muchos trabajos usan metas, RNF, escenarios y modelos de roles. Adicionalmente, existe un interés en el campo de SMA por el uso de patrones para modelar desde un nivel abstracto (por ejemplo social) a uno más detallado y orientado a arquitectura.

Además, existe una carencia de investigación para la trazabilidad entre artefactos producidos durante el desarrollo de SMA. La trazabilidad pre-requisitos solo se aplicó en un trabajo (Paraense, Gudwin, & Gonçalves, 2007). La aplicación de mejores mecanismos y herramientas para tratar con la pre- y post- trazabilidad en las metodologías SMA podrían ayudar a identificar las necesidades del usuario, mejorar su comprensión del sistema, y mejorar la calidad global del software desarrollado.

Por otra parte, los resultados también muestran que existe una necesidad de más estudios empíricos para validar las propuestas. Los estudios empíricos son bloques necesarios para construir evidencias y determinar que técnicas son mejores en ciertas situaciones.

Finalmente, los resultados de este trabajo se tuvieran en cuenta para el desarrollo de la propuesta de RE4Gaia: en la definición de los modelos, estructura y metodología. Primeramente se consideró integrar técnicas de modelado de requisitos en la metodología Gaia, para dar soporte a la adquisición y modelado de especificaciones de requisitos en SMA y el posterior análisis de estas.

Otro resultado interesante que influyó en nuestra propuesta fue el amplio uso de la notación i*. Se barajó la integración de esta notación, pero se comprobó mediante la experimentación con un caso de estudio, basado en la gestión de una conferencia, y se detectó que la complejidad aumentaba drásticamente, complicándose el aprendizaje de la metodología. Finalmente, se optó por realizar una solución basada en UML2 dado su alto soporte industrial mediante herramientas y su facilidad de integración con el resto de estándares propuesto por la OMG.

Otro resultado que influyó en nuestra propuesta fue el uso intensivo de roles dado que facilita la visión de un sistema como con conjunto de componentes que interactúan entre sí. En consecuencia, se optó por incluir roles a un nivel temprano en el modelado de requisitos, al ser una notación muy útil para este tipo de sistemas.

También se hizo especialmente el soporte a la trazabilidad con el objetivo mejorar el entendimiento de las necesidades del sistema, facilitar la mantenibilidad de los artefactos producidos y mejorar la calidad del sistema desarrollado. Además se incluyó esta propuesta en un entorno de Desarrollo Dirigido por Modelos para potenciar la usabilidad y mantenimiento de las especificaciones.

Por último, un tercio de las propuestas para tratar con los requisitos en SMA no proporcionaban ningún tipo de validación del método. En nuestro caso decidimos realizar una validación inicial mediante un caso de estudio. Se ha planteado como trabajo futuro la validación del proceso propuesto con una serie de experimentos que permitirán analizar la eficacia y entendibilidad del método y la retroalimentación para la mejora del mismo.

4 RE4Gaia

En esta sección se presentará la propuesta para dar soporte a la Ingeniería de Requisitos en la metodología Gaia.

4.1 Introducción

El proceso de *Ingeniería de Requisitos* (IR) es reconocido como el más crítico para el desarrollo del software. Los errores durante este proceso podrían tener efectos negativos en las fases posteriores del desarrollo, y consecuentemente influir en la calidad del resto del software desarrollado.

Por otra parte, un Sistema Multi-Agente es un tipo especifico de sistema compuesto por múltiples agentes que interactúan entre sí para resolver problemas que serían difíciles (o imposibles) de resolver en un sistema de una agente individual o monolítico. En los últimos años se han propuesto muchas metodologías orientadas a agentes para dar soporte al desarrollo de este tipo de sistemas, siguiente diferentes enfoques: orientados a metas, orientados a objetos, etc. Desafortunadamente, muchas de estas propuestas se enfocan solo en el análisis, diseño o implementación de SMA.

Posiblemente la metodología de desarrollo de sistemas Multi-Agente más usada sea Gaia (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003). Gaia está basada en el uso de metáforas organizacionales y se cimenta en el uso de la metáfora organizacional, viendo los SMA como organizaciones computacionales consistentes en múltiples agentes que interactúan entre sí. De acuerdo con Gaia, un SMA se ve como una composición de un número de agentes autónomos que interactúan entre en una sociedad organizada en la que cada agente puede efectuar uno o más roles (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003).

Cuando se aplica Gaia, el analista partirá de conceptos abstractos hasta acabar trabajando con roles e interacciones concretas. En Gaia se sugiere que los analistas y diseñadores trabajen con Gaia en un proceso de desarrollo donde se incrementa el nivel de detalle de los modelos del sistema que se va a construir. Sin embargo, en Gaia los requisitos son simplemente *sentencias*, independientes del paradigma utilizado para en análisis y diseño; en lugar de emplear un modelo relevante en un SMA, que sea orientado a la adquisición de requisitos.

Otras desventaja en Gaia, desde nuestro punto de vista, es la falta de una trazabilidad explicita desde los requisitos a artefactos producidos en el desarrollo de SMA. Un mejor mecanismo de trazabilidad ayudaría a conocer las necesidades del usuario, aumentando el entendimiento del sistema, y facilitando la mantenibilidad de los artefactos producidos, y mejoraría la calidad global del software desarrollado (Blanes, Insfrán, & Abrahão, International Symposium on Distributed Computing and Artificial Intelligence, 2009).

En esta sección, se presentará RE4Gaia, un enfoque de modelado de requisitos para el desarrollo de SMA extendiendo la metodología Gaia. Este enfoque se basa en el uso de estructuras organizacionales como medio para adquirir y entender los roles y funciones necesitadas, en el contexto de una organización, necesarias para en análisis y diseño utilizando Gaia.

Este capítulo se organizara de la siguiente manera. La sección 4.2 se hará una descripción global de RE4Gaia, dentro de esta parte se incluyen dos sub-secciones. La primera, la sub-sección 4.2.1 describirá la fase de modelado de requisitos en REGaia. La segunda, la sub-sección 4.2.2

describe la fase de análisis de RE4Gaia. Finalmente, la sección 4.3 muestra las conclusiones obtenidas de este capítulo.

4.2 RE4Gaia

El enfoque de modelado de requisitos propuesto se basa en el uso de la estructura organizacional con el objetivo de adquirir y entender adecuadamente los roles y funciones asociadas requeridas, en el contexto de una organización, paso previo al análisis y diseño del SMA utilizando Gaia. Esto incluye un conjunto de técnicas para adquirir y representar los requisitos de software de un SMA. Específicamente, nuestro enfoque provee:

i. Una fase de modelado de requisitos.
ii. Un proceso de análisis de requisitos.

La figura 4-1 muestra una visión de los modelos y sus respectivas relaciones en RE4Gaia.

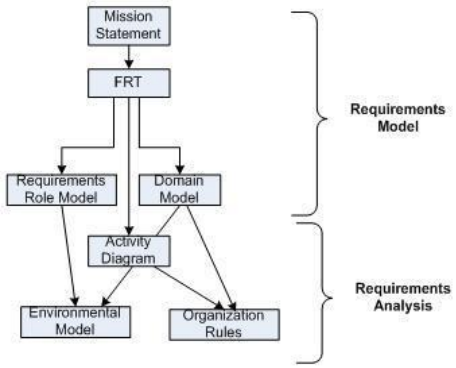


Figura 4-1 Modelos de RE4Gaia y relaciones entre ellos

4.2.1 Fase de Modelado de Requisitos

El objetivo del Modelo de Requisitos es adquirir y representar los requisitos de software. Esta fase comienza con la definición de la Misión del Sistema. A continuación se define el Árbol de Refinamiento de Funciones (ARF). Una vez construido el ARF, la información obtenida de este servirá para construir el Modelo de Roles de la fase de Requisitos (MRR) y el Modelo de Dominio (MD) tal como muestra la figura 4-1. Finalmente, las salidas de los modelos ARF, MRR y DM se utilizarán como entrada para la construcción de los modelos de la fase de Análisis de Requisitos.

La Misión del Sistema establece las metas globales de la organización. El ARF ayuda a determinar las sub-organizaciones que componen la organización global y sus roles participantes. El MRR se utiliza para detectar relaciones de herencia entre roles y razonar sobre sus relaciones estructurales, con el objetivo de detectar posibles inconsistencias. Finalmente, en MD se utiliza para detectar las entidades que podrían utilizarse como recursos de la organización.

4.2.1.1 Misión del Sistema

La **Misión del Sistema (MS)** es el mayor servicio (la meta global) que el sistema a desarrollar proveerá a su entorno (Insfran, 2003). Está escrita en lenguaje natural, con una extensión recomendada de uno o dos párrafos. El propósito de la MS es determinar de una forma clara y concisa lo que el sistema debe hacer (y si se considera necesario, lo que el sistema no debería hacer).

4.2.1.2 Árbol de Refinamiento de Funciones

Una de las mayores ventajas de Gaia, es el uso de la metáfora organizacional. Gaia sugiere dividir el sistema en sub-organizaciones. Esta división tiene la ventaja de la aplicación de los principios de modularidad y encapsulación del software. Para obtener ventaja de la metáfora organizacional, usaremos la técnica llamada Árbol de Refinamiento de Funciones (ARF) para agrupa las diferentes áreas en las que se descompone la organización.

El ARF se utiliza para representar un jerarquía, basada en una descomposición de las funciones de negocio independiente del la estructura del sistema de software futuro.

Primeramente, ponemos en el nodo *raíz* del árbol la Misión del Sistema. La Misión del Sistema se refina sucesivamente para identificar las funciones del sistema, que se representarán como nodos *hoja* en el ARF. Este proceso, pueden distinguirse diferentes niveles. Los nodos entre la raíz y las hojas se llaman nodos *intermedios*.

Podemos distinguir entre dos niveles de nodos intermedios en el ARF. En el primer nivel de los nodos intermedios, se encuentran los *Grupos Funciones* llamados como **sub-organizaciones**. En este contexto entenderemos sub-organización como una parte del sistema que está orientada a conseguir un objetivo en el sistema y que interactúa débilmente con otras partes del sistema (bajo acoplamiento).

Las sub-organizaciones del primer nivel de los nodos intermedios, se descomponen en diferentes **roles**. Los roles se sitúan en el segundo nivel de los nodos intermedios. Entenderemos como *rol* en este contexto como la representación de una entidad abstracta que provee (múltiples) funciones del sistema. Una función es una tarea llevaba a cabo por un rol en la organización, independientemente de la necesidad de colaborar con otros roles o no.

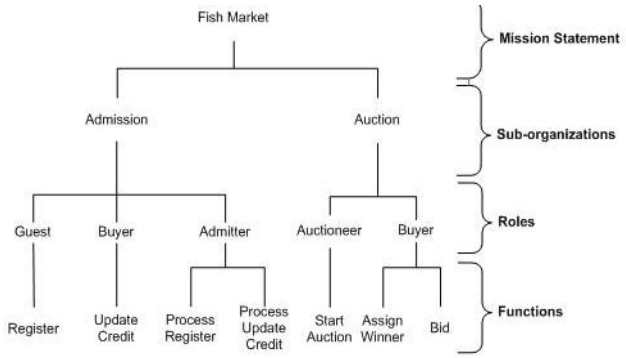


Figura 4-2 Ejemplo de un Árbol de Refinamiento de Funciones

La figura 4-2 muestra un extracto de un ARF para un sistema que gestiona un mercado de venta de pescado, el cual se detalla en detalle en la sección 5. En el ejemplo mostrado en la figura, el sistema se divide en dos sub-organizaciones: *Admission* y *Auction*. La sub-organización *Admission* está compuesta por los roles: *Guest*, *Buyer* y *Admitter*. El rol *Guest* realiza la función *Register*, el rol *Buyer* la función *Update Credit* y el rol *Admitter* realiza las funciones *Process Register* y *Process Update Credit*. Por otra parte, la sub-organización *Auction* está compuesta por los roles: *Auctioneer* y *Buyer*. El rol *Auctioneer* puede realizar la función *Start Auction*, el rol *Buyer* puede realizar las funciones *Assign Winner* y *Bid*.

4.2.1.3 Modelo de Roles de Requisitos

El Modelo de Roles de Requisitos describe los roles que pertenecen a las sub-organizaciones del ARF. Entenderemos como rol a una entidad abstracta que provee (múltiples) **funciones** al sistema. El propósito de este modelo es representar los diferentes roles descubiertos en la organización y razonar sobre sus relaciones particulares. Particularmente se necesita identificar las propiedades comunes entre los roles para crear una jerarquía de roles usando relaciones de *herencia*. Para representar gráficamente esta información, se empleará el Diagrama de Casos de Usos de UML mostrando los roles como actores, etiquetándolos con el estereotipo <<role>>, y las relaciones entre ellos. En la figura 7-1 se muestra un ejemplo de este tipo de diagrama.

4.2.1.4 Modelo de Dominio

El Modelo de Dominio muestra las entidades identificadas en el dominio del problema. El propósito es identificar los conceptos claves y sus relaciones, representando una primera visión estructural. Primero, se identifican las **entidades** del dominio relevantes para el dominio de la aplicación. Una entidad puede ser tanto una entidad física como un concepto. Estas entidades son vistas desde el punto de vista del dominio de la aplicación, de modo que en este nivel se evitaran detalles de implementación. Adicionalmente, se representarán **asociaciones** entre las entidades del dominio. Estas asociaciones se pueden anotar con las multiplicidades: *cero, una, muchas* o una *constante*. Finalmente, podemos refinar el modelo con relaciones de **herencia**.

Para representar gráficamente esta información, se emplearán Diagramas de Clases de UML, mostrando únicamente las entidades como clases y sus relaciones entre ellas (asociaciones y herencias). Un ejemplo de este tipo de diagrama, se muestra en la figura 7-2.

4.2.2 Fase de Análisis de Requisitos

El Proceso de Análisis de Requisitos comienza tomando como entrada las funciones identificadas en el ARF y las descompone en *tareas* y *protocolos*. Usaremos para este propósito el Diagrama de Actividad de UML. Usaremos el Diagrama de Actividad pare entender el flojo interno de un rol con el propósito de determinar sus responsabilidades del modelo de roles de Gaia. Una vez que se han identificado las actividades y tareas, se identifican los recursos (Modelo de Entorno), refinando las entidades identificadas en el Modelo de Dominio en *recursos* y *permisos*. Una vez hemos identificado las tareas y protocolos de roles y los recursos a los que acceden, se definen las Reglas Organizacionales para definir el comportamiento y restricciones de la dinámica de la organización, vista como un todo.

4.2.2.1 Diagrama de Actividad

El Diagrama de Actividad específica construcciones dinámicas usando actividades y acciones. Este diagrama muestra una secuencia de pasos que muestran el flujo de datos necesarios para realizar las funciones identificadas. El Diagrama de Actividad se considera una extensión de los *flowchars* tradicionales. En los modelos de *flowchart* se modela el flujo paso a paso. Se utilizan típicamente en el modelado de algoritmos. Sin embargo, en el Diagrama de Actividad se modela de forma secuencial y concurrente el flujo de controles. Estas características hacen que el Diagrama de Actividad sea una buena opción para modelar organizaciones humanas donde se ejecutan operaciones de forma concurrente.

En RE4Gaia, utilizaremos el Diagrama de Actividad para asistir en análisis e identificación de dependencias entre roles. Una representación para el flujo de tareas podría ser útil para entender

el flujo lógico de un rol; ayudando a identificar cuando un rol necesita colaborar con otros para realizar una tarea.

Respecto a los elementos que utilizaremos en el Diagrama de Actividad, comenzaremos con el N**odo Inicial**. Se representará como un círculo sólido con una flecha que parte de él. Los nodos iniciales no tienen aristas entrantes. En oposición al nodo inicial, el **Nodo Final** indica que el flujo de actividad se detiene. Se representa con un círculo sólido con un borde. Una **Acción** representa una actividad que ocurre en el diagrama de actividad. Se representa con un rectángulo redondeando. El flujo entre acciones se modela con el elemento **Flujo de Control**. El flujo de control que indica el comienzo de una actividad después de la finalización de una actividad previa. Se representa gráficamente en el diagrama con una flecha.

La concurrencia se representa con los elementos de bifurcación (*fork*) y unión (*join*). La **Bifurcación** se representa con una barra negra con un flujo de entra y varios flujos que parten de él. Este elemento indica el principio de una actividad paralela. En oposición a esta, la **Unión** indica el fin de un proceso paralelo. Se representa con una barra negra con varios flujos de entrada y uno de salida. Los **Nodos de Decisión** se representan con un rombo con un flujo de entrada y varios de salida. Las aristas salientes son **Flujos de Control**, este tipo de flujos tiene una guarda que se utiliza para decidir entre cuales de las opciones irá el flujo de salida.

Una **Partición de Actividad**, es un tipo de grupo de actividades usado para identificar acciones que tienen unas características en común. En RE4Gaia se utilizará para identificar el conjunto de actividades y protocolos que lleva a cabo un determinado rol. Gráficamente representaremos una partición con dos líneas verticales y paralelas entre sí, y una caja que contendrá una etiqueta con el nombre de la partición. Los nodos de actividad y las aristas situadas entre estas líneas, se considera que están contenidas dentro de la partición.

Con el objetivo de representar los *protocolos*, utilizaremos los elementos Enviar Señal de Acción (*Send Signal Action*) y Aceptar Evento Acción (*Accept Event Action*). El evento ***Send Signal Action***, indica el comienzo de un protocolo. Este elemento se representa gráficamente con un pentágono convexo. Adicionalmente, se recomienda utilizar un texto dentro del pentágono con el nombre del protocolo que se inicia y a continuación el nombre del rol que responde entre paréntesis. Para modelar la respuesta a un protocolo, se empleará el elemento ***Accept Event Action***. Este elemento modela la espera de la ocurrencia de un evento que active el protocolo bajo unas condiciones específicas. La representación gráfica es mediante un pentágono cóncavo. Se sugiere indicar dentro de este pentágono el nombre del protocolo.

A continuación se listarán unas guías para construir correctamente los Diagramas de Actividad:

- Es necesario construir al menos un Diagrama de Actividad para cada sub-organización identificada en el ARF. Dependiendo de la complejidad del problema, podemos construir más de un diagrama por sub-organización.
- Se sugiere construir un Diagrama de Actividad para cada rol que sea proactivos, entendiéndose como proactivos aquellos roles que inicien otros protocolos.
- Cada Partición de un Diagrama de Actividad modela el flujo lógico de un rol. Se sugiere que la partición de la izquierda se utilice para modelar el comportamiento de un rol que inicia una partición. El resto de particiones representan los otros roles con los que interactúa el rol iniciador.

- Si se identifica que un rol necesita interactuar con más roles, entonces creamos una nueva Partición en el Diagrama de Actividad. Si el rol nuevo tiene la habilidad de iniciar protocolos, se sugiere crear un nuevo Diagrama de Actividad para ese rol.

4.2.2.2 Modelo de Entorno

El Modelo de Entorno es un conjunto de tuplas con la siguiente estructura: role, recurso y permiso. Para cada rol del Diagrama de Roles, establecemos los **recursos** a los que puede acceder legítimamente el rol. La información extraída del Diagrama de Actividad ayudará al analista a identificar a que recursos se acceden. Finalmente se establecerán los **permisos** con los que el rol accede a los recursos. Distinguiremos entre tres tipos de permisos, de una forma similar a la metodología Gaia: *leer, escribir y consumir*.

4.2.2.3 Reglas Organizacionales

El último paso en el Análisis de Requisitos es definir las Reglas Organizacionales. Se escribirán en lenguaje natural con el objetivo de identificar y representar restricciones generales en el comportamiento de la Organización. Estas reglas pueden verse como responsabilidades sobre la organización vista como un todo. Pueden ser de dos tipos:

i. Propiedades Dinámicas, definen como debería de comportarse la Organización.
ii. Restricciones de la Organización, definen invariantes que la organización debe respetar en cada instante de tiempo.

4.3 Conclusiones

En este capítulo se ha presentado RE4Gaia, un enfoque de modelado de requisitos para SMA propuesto para extender la metodología Gaia. Este enfoque se basa en la estructura organizacional para adquirir y entender adecuadamente los roles y funciones asociadas en el contexto de una organización anteriormente al análisis de diseño usando Gaia.

Consideramos que este enfoque cubre una carencia en el desarrollo de SMA, proveyendo un enfoque sistemático para tratar con los requisitos estableciendo un mecanismo de trazabilidad que a ayudará a identificar las necesidades del usuario, mejorar su entendimiento del sistema y facilitar la mantenibilidad de los artefactos producidos, mejorando la calidad global del software desarrollado.

5 Desarrollo de Software Dirigido por Modelos y RE4Gaia

En esta sección se definirán los meta-modelos tanto de RE4Gaia como el de la fase de análisis de Gaia. Además se comentarán las transformaciones de modelos definidas entre ellos, con el objetivo de dar soporte al Desarrollo de Software Dirigido por Modelos. Por último se presentarán las conclusiones del capítulo.

5.1 Introducción

El Desarrollo de Software Dirigido por Modelos es una propuesta con el objetivo de maximizar la productividad del software potenciando aspectos como la reusabilidad, interoperabilidad y la mejora adaptación del sistema a cambios tecnológicos. Una de las propuestas más populares de DSDM es la arquitectura de desarrollo de software dirigida por modelos (MDA), la cual se apoya en el conjunto de estándares propuestos por la OMG (Object Management Group, 2009). En esta arquitectura juegan un papel clave las definiciones de modelos a partir de meta-modelos y las transformaciones automáticas entre ellos. El estándar de la OMG llamado MOF (Object Management Group, 2008) propone una arquitectura de modelado basada en cuatro capas. En el nivel superior o M3 residen los meta-metamodelos que se utilizan como lenguajes para definir lenguajes de modelado. En el nivel M2 residen los meta-modelos que son modelos que se utilizan para describir modelos. En el nivel M1 se encuentran los modelos, que son instancias de los meta-modelos del nivel M2. Finalmente en el nivel M0 se encuentran las instancias de estos modelos. Las transformaciones entre modelos se definirán a nivel M2 mediante el lenguaje de transformaciones *QVT Relations*. QVT Relations es una implementación declarativa del estándar QVT (Object Management Group, 2008), con la principal ventaja de que tiene soporte a transformaciones bidireccionales.

En este trabajo se definirán los meta-modelos (nivel M2 de la arquitectura MOF) para RE4Gaia y la fase de análisis de la metodología Gaia. El apartado 5.2 contiene una descripción del meta-modelo RE4Gaia. En el apartado 5.3 reside la descripción del meta-modelo Gaia. Por otra parte, en el apartado 5.4 se comentan las reglas de transformación definidas entre los dos meta-modelos, escritas en el lenguaje *QVT Relations*. Finalmente, en el apartado 5.5 se comentan las conclusiones de este capítulo.

5.2 Meta-modelo RE4Gaia

La figura 5-1 muestra una visión global del meta-modelo RE4Gaia. Cada uno de los paquetes contiene los elementos necesarios para construir un modelo en RE4Gaia.

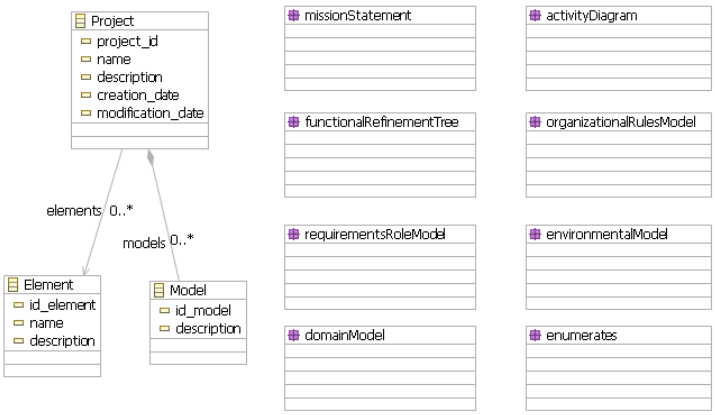


Figura 5-1 Clases Principales del meta-modelo RE4Gaia y paquetes incluidos

5.2.1 Paquete *mission Statement*

El paquete *misión Statement*, contiene una meta-clase con la información de la misión del Sistema. En la figura 5-2 se ve la representación gráfica de esta.

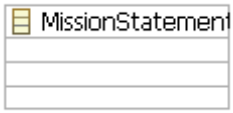


Figura 5-2 Meta-clase *Mission Statement*

5.2.2 Paquete functional Refinement Tree

En este paquete se incluyen los elementos para representar un Árbol de Refinamiento de Funciones. La figura 5-3 muestra una vista de las clases que componen este paquete.

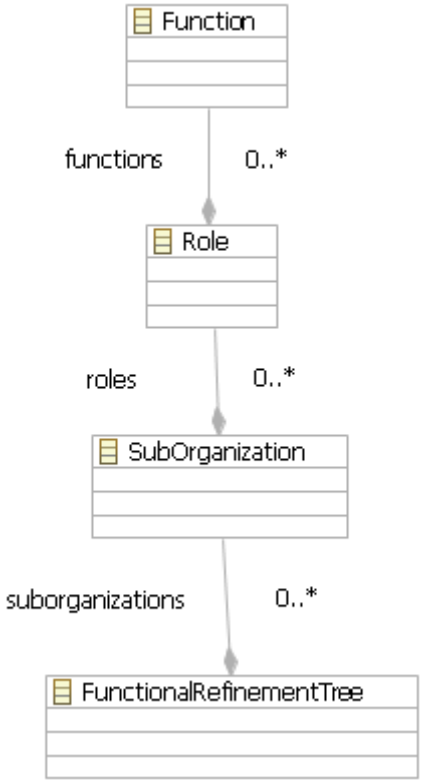


Figura 5-3 Meta-clases y relaciones dentro del paquete *Funcional Refinement Tree*

La meta-clase *Functional Refinement Tree* actúa como contenedor de los elementos de este modelo. La relación de agregación entre las meta-clases *Functional Refinement Tree* y *Sub-Organization* indica que un FRT está formado por múltiples sub-organizaciones. La meta-clase Role represente al conjunto de Roles que contiene una sub-organización. La relación de agregación entre *Sub-Organization* y *Role* indica esto. Por último la meta-clase *Function* representa una función llevaba a cabo por un rol. Un rol realiza un conjunto de funciones, representado en el meta-modelo como una relación de agregación entre las meta-clases Role y *Function*, tal como se muestra en la figura.

5.2.3 Paquete requirements Role Model

Este paquete contiene las meta-clases para modelar el Modelos de Roles de la fase de Requisitos. La figura 5-4 muestra una vista de las meta-clases y las relaciones entre ellas que componen este paquete.

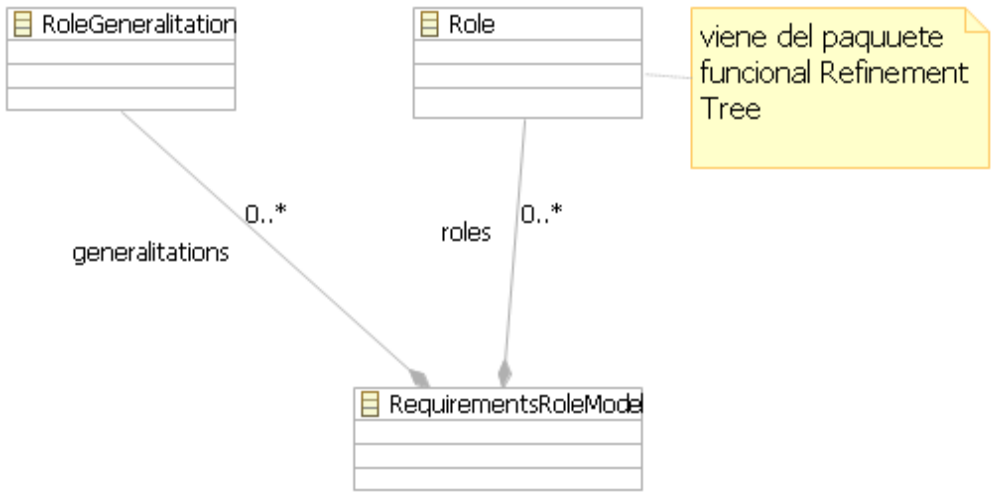


Figura 5-4 Meta-clases del paquete *Requirements Role Model*

La meta-clase *Requirements Role Model* especifica un Modelos de Roles. Este contiene un conjunto de Generalizaciones, representado mediante una agregación entre las meta-clases *Requirements Role Model* y *Role Generalizations*. La clase *Role* se incluyó en el paquete *funcional Refinement Tree*, al participar también en el ARF.

5.2.4 Paquete domain Model

Este paquete contiene las meta-clases y relaciones entre ellas para modelar un Modelo de Dominio. La figura 5-5 muestra una vista de estos elementos.

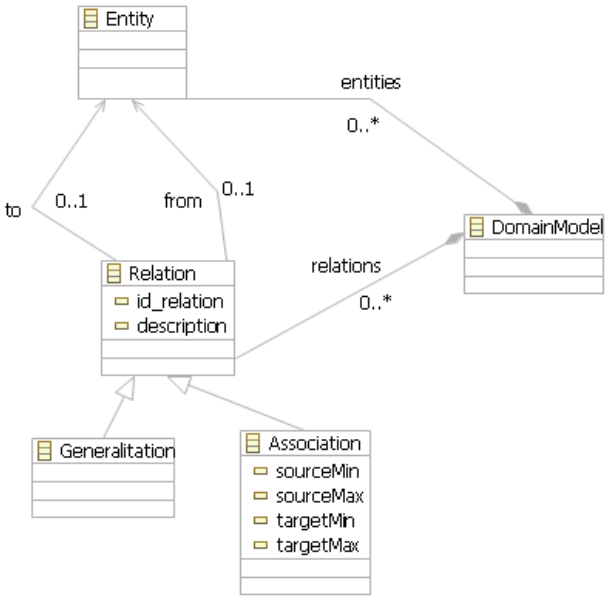


Figura 5-5 Meta-clases y relaciones entre ellas del paquete *Domain Model*

La meta-clase *Domain Model* actúa como contenedor. Un modelo de dominio está compuesto por entidades y relaciones, representados respectivamente mediante las meta-clases *Entity* y *Relation*. *Relation* es una meta-clase abstracta que representa el concepto de relación entre dos entidades. Una relación se puede especializar en una generalización o una asociación, modelados con las meta-clases *Generalization* y *Association*.

5.2.5 Paquete Activity Diagram

En este paquete se incluyen las meta-clases y relaciones entre ellas necesarias para especificar todos los elementos de un Diagrama de Actividad en la fase de Análisis de Requisitos en RE4Gaia. La figura 5-6 muestra los elementos del paquete.

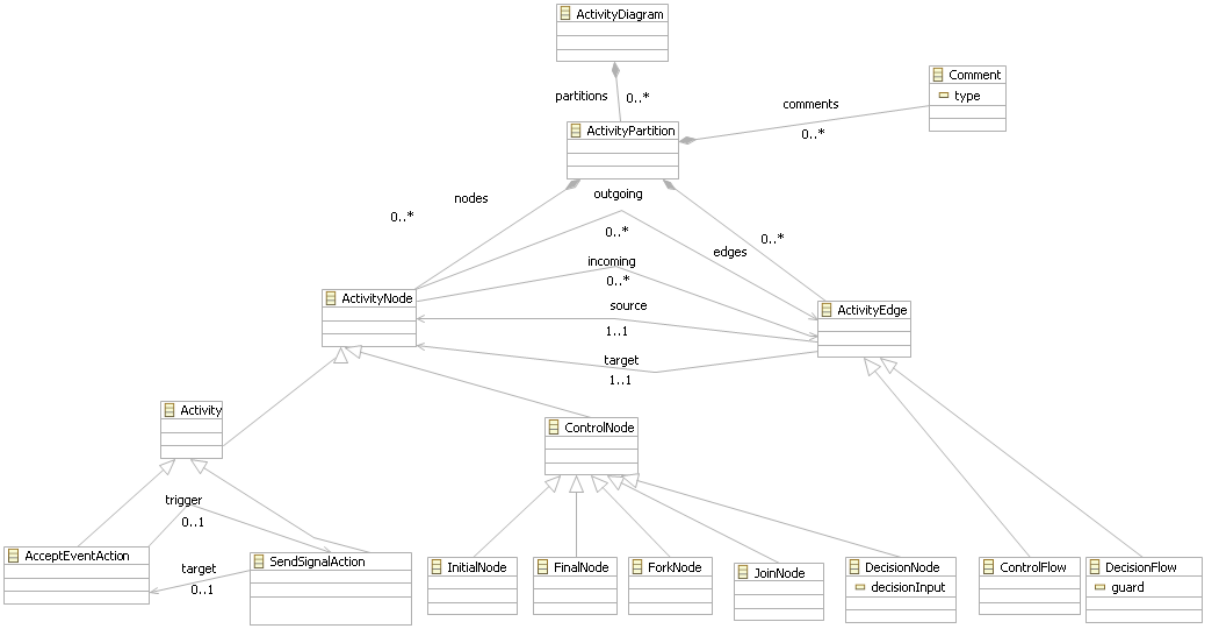


Figura 5-6 Meta-clases y relaciones entre ellas del paquete *Activity Diagram*

La meta-clase *Activity Diagram* es el contenedor del diagrama de actividad. Está compuesto de diferentes particiones, especificadas en la meta-clase *Activity Partition*. Una partición a su vez se compone de nodos, aristas y comentarios. Esta relación se modela en el diagrama con relaciones de agregación que parten de la meta-clase *Activity Partition* a las meta-clases *Activity Node*, *Activity Edge* y *Comment*. Un *Activity Node* es una meta-clase abstracta, que puede especializarse en un *Activity* o un *Control Node*.

5.2.6 Paquete Organizational Rules Model

Este paquete contiene los elementos necesarios para representar un Modelo de Reglas Organizacionales. La figura 5-7 muestra las meta-clases y sus relaciones que la componen.

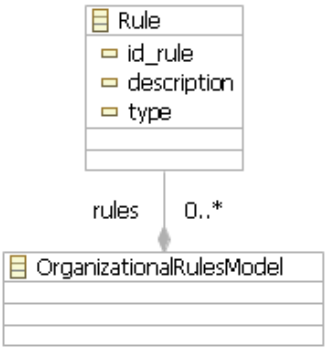


Figura 5-7 Meta-clases y relaciones entre ellas del paquete *Organizational Rules Model*

La meta-clase *OrganizationRulesModel* es el contenedor de un conjunto de reglas que forman el modelo. Esto se representa mediante la agregación de la meta-clase *OrganizationalRulesModel* a la meta-clase *Rule*.

5.2.7 Paquete Environmental Model

Este paquete contiene los elementos necesarios para especificar un Modelo de Entorno. Aquí se muestran los recursos del sistema y los permisos mediante los cuales pueden ser accedidos por los roles. La figura 5-8 muestra una vista de las meta-clases y las relaciones entre ellas contenidas en el paquete.

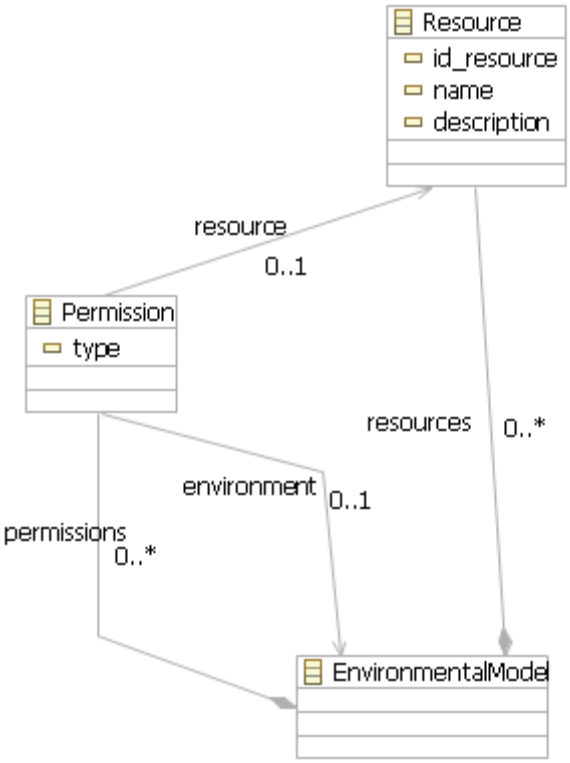


Figura 5-8 Meta-clases y relaciones dentro del paquete *Environmental Model*

La meta-clase *EnvironmentalModel* es el contenedor de los elementos del Modelo de Entorno. Este modelo contiene un conjunto de Recursos, modelados con la meta-clase *Resource*. Los

permisos permiten a un rol acceder a un determinado recurso. Este comportamiento se modela con la meta-clase *Permission*.

5.2.8 Paquete *Enumerates*

Este paquete contiene las Enumeraciones del meta-modelo RE4Gaia. La figura 5-9 muestra una vista de las enumeraciones contenidas en el paquete.

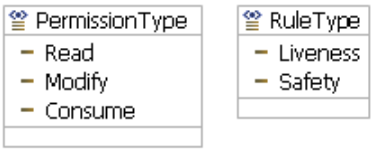


Figura 5-9 Paquete *Enumerates*

5.3 Meta-modelo Gaia

Esta meta-modelo incluye todos los elementos necesarios para especificar los modelos de análisis en la metodología Gaia. La figura 5-10 muestra una visión global de todos los elementos del meta-modelo.

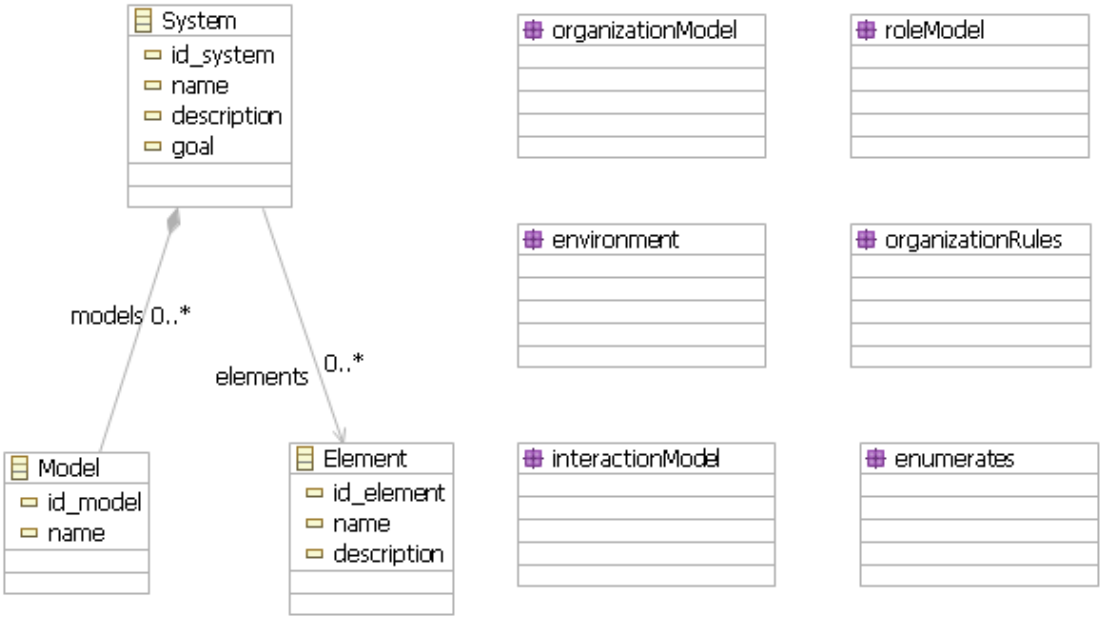


Figura 5-10 Elementos del meta-modelo metaGaia

La meta-clase *System* representa a un contenedor de todos los elementos del sistema.

5.3.1 Paquete *SubOrganizations*

En este paquete están contenidos los elementos para representar el conjunto se sub-organizaciones que forman un sistema en Gaia. La figura 5-11 muestra las meta-clases y sus relaciones en este paquete.

Figura 5-11 Meta-clases y relaciones del paquete *SubOrganizations*

La meta-clase subOrganizations es el contenedor del conjunto de sub-organizaciones, representadas estas últimas con la meta-clase Suborganization.

5.3.2 Paquete *environment*

Paquete que contiene los elementos necesarios para representar un Modelo de Entorno. En la figura 5-12 mostramos las meta-clases y las relaciones entre ellas de este paquete.

Figura 5-12 Meta-clases del paquete E*nvironment*

5.3.3 Paquete roleModel

En este paquete están contenidas las meta-clases y sus respectivas relaciones entre ellas para especificar un Modelo de Roles. La figura 5-13 muestra los elementos de este paquete.

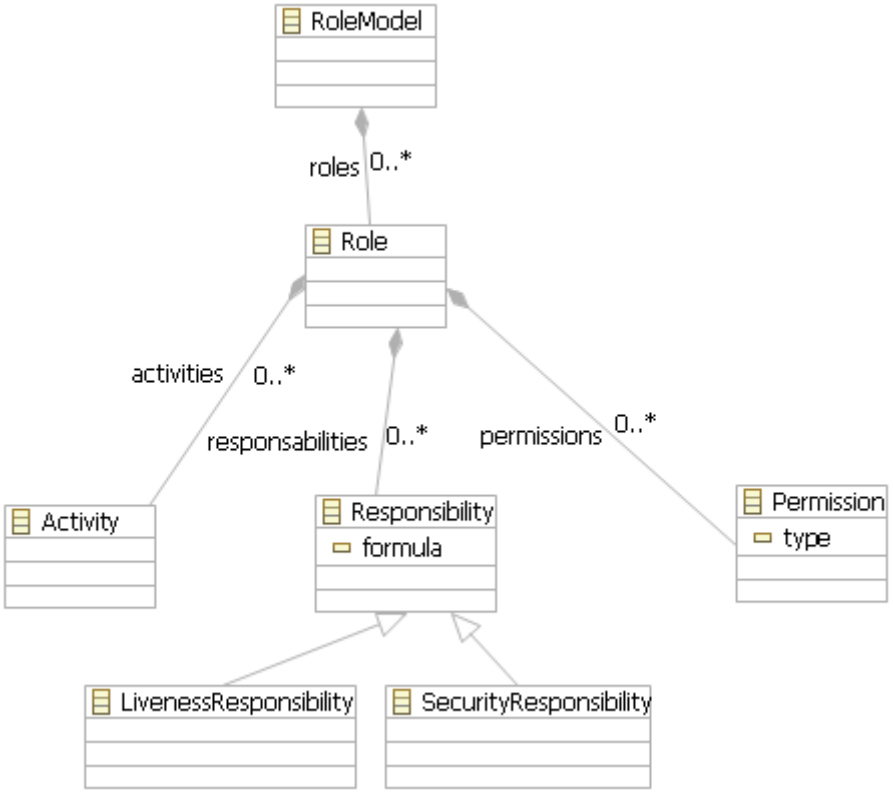


Figura 5-13 Meta-clases del paquete *Role Model*

La meta-clase RoleModel contiene el conjunto de roles que participan en el sistema. Un rol a su vez se compone de actividades (meta-clase *Activity*), responsabilidades (meta-clase *Responsibility*) y permisos (meta-clase *Permission*).

5.3.4 Paquete interactionModel

Contiene los elementos de un Modelo de Interacciones en Gaia. La figura 5-14 muestra una vista de las meta-clases y relaciones entre ellas de este paquete.

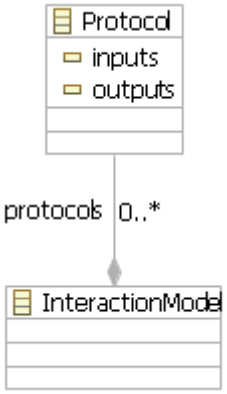


Figura 5-14 Elementos del paquete *Interaction Model*

5.3.5 Paquete Organizational Rules

Contiene los elementos para construir un Modelo de Reglas Organizacionales. La figura 5-15 muestra el contenido de este paquete.

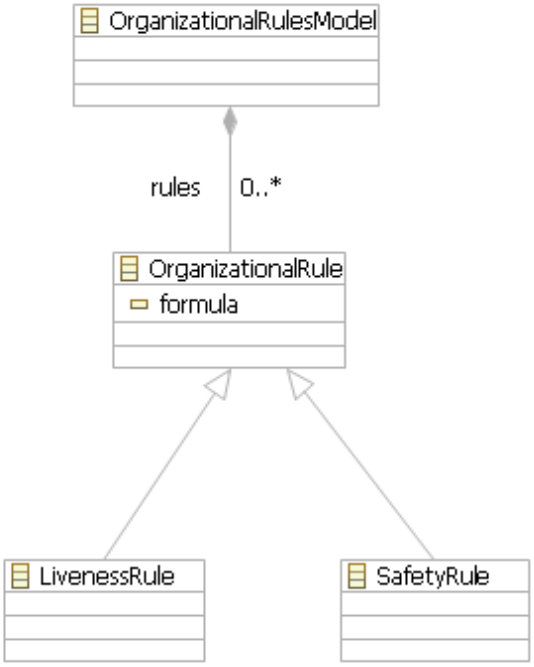


Figura 5-15 Elementos del paquete organizationalRules

5.4 Reglas de Transformación

En este se detalla cómo se han construido las reglas de transformación entre los meta-modelos expuestos en las secciones anteriores. Primero se listarán, en lenguaje natural, las reglas de correspondencia entre los elementos de los meta-modelos. Estas reglas se utilizarán para escribir las reglas de transformación en el lenguaje QVT. En los siguientes puntos se explicará cómo se han utilizado los principales fundamentos del lenguaje QVT (Object Management Group, 2008) para definir las reglas de transformación construidas para realizar transformaciones desde modelos RE4Gaia a modelos de análisis en Gaia. La especificación completa de las reglas de transformación generadas se encuentra en el Anexo C.

5.4.1 Reglas de Correspondencia

En este punto se describirán las reglas de trazabilidad entre modelos. Se describirá para cada elemento del modelo origen, cuál será su elemento del modelo destino correspondiente. Esta información se utilizará para escribir las reglas mediante el lenguaje *QVT Relations*. A continuación se detallas las reglas de correspondencia:

- Un *Project* en RE4Gaia se convierte en un *System* en Gaia.
- El atributo *description* de *MissionStatement* en RE4Gaia se convierte en un atributo *goal* en la meta-clase *System* de Gaia.
- Por cada *FunctionalRefinementTree* en el origen se crea una *Organization* en el destino.
- Una clase *SubOrganization* del modelo origen se convierte en una clase *SubOrganization* en el modelo destino.
- A cada *RequirementsRoleModel* le corresponde un RoleModel.
- A un *Role* en el origen, le corresponde un *Role* en el destino.

- Para cada elemento *SendSignalAction* en el origen, le corresponde un *Protocol* en el destino. Para cada rol que invoca el *SendSignalAction* en el diagrama de actividad del modelo origen; en el Protocolo del modelo destino, se referencia mediante a este rol mediante el atributo *initiator*. Para cada rol asociado en el atributo *target* en un *SendSignalAction* en el modelo origen, en el Protocolo destino se construye una referencia mediante el atributo *partner* a ese rol.
- Una *Activity* en el modelo origen, se convierte en una *Activity* en el modelo destino.
- Un *Comment*, que tiene como *type* el literal *Liveness*, se convierte en una *LivenessResponsibility*.
- Un *Comment*, que tiene como *type* el literal *Safety*, se convierte en una *SecurityResponsibility*.
- Un EnvironmentalModel se convierte en un Environment en el destino.
- Un *Resource* del modelo origen, se convierte en un *Resource* en el destino.
- Un *Permission* del modelo origen se convierte en un *Permission* en el destino. El tipo del destino (*read, write, consume*) en el origen será el correspondiente al modelo origen.
- A un OrganizationalRulesModel le corresponde un OrganizationalRulesModel.
- Un *Rule* del modelo origen puede convertirse en un *LivenessRul*e o un *SafetyRule*. El tipo depende del valor seleccionado en el atributo *type*.

5.4.2 Definición de Transformaciones

En el lenguaje *Relations*, se definen transformaciones entre modelos candidatos mediante un conjunto de relaciones que se deben cumplir para que la transformación se cumpla exitosamente. Un modelo candidato es un modelo que conforma un tipo de modelo. Un tipo de modelo es una especificación del tipo de elementos que contiene un modelo que lo conforma. Los modelos candidatos poseen nombre, y los tipos de elementos que pueden contener se restringen a aquellos que están dentro del conjunto de paquetes referenciados.

Nuestras transformaciones están comprendidas dentro de una declaración:

```
transformation RE4GaiaToGaia(source: metaRE4Gaia, target: metaGaia){
      …
}
```

En esta declaración, llamada RE4Gaia, existen dos tipos de modelos candidatos: "*metaRE4Gaia*" y "*metaGaia*". El modelo llamado "*source*" declara un paquete metaRE4Gaia como meta-modelo, mientras que el modelo "*target*" declara un paquete metaGaia como meta-modelo.

Respecto a la dirección de la transformación, consideraremos que las transformaciones parten del modelo "*source*" al modelo "*target*". El modelo "*target*" se considerará vacío inicialmente. La ejecución de la transformación procede de la siguiente manera, primero se comprueba que relaciones se cumplen y en cuales la comprobación falla. Se intentará que se cumplan las relaciones creando, borrando o modificando únicamente en el modelo origen para forzar a que se cumpla la relación.

5.4.3 Definición de Claves

Respecto a los objetos que creamos en el modelo destino, en algunos casos desearemos que no se dupliquen objetos que ya existan. En los casos que simplemente queremos creamos

actualizarlos objetos existentes recurriremos a una propiedad permitida en MOF para nombrar identificadores en una clase. Sin embargo en muchos meta-modelos no es suficiente identificar los objetos. Para esos casos el lenguaje *Relations* introduce el concepto de *Key*, el cual define un conjunto propiedades en una clase que identifican inequívocamente una instancia del objeto en un Modelo. Una clase puede tener múltiples *Keys*, de forma similar a la base de datos relacionales.

```
/* Key definition */
key metaRE4Gaia::functionalRefinementTree::Role {name};
key metaRE4Gaia::interactionModel::Protocol {name};
key metaRE4Gaia::environment::Resource {name} ;
key metaRE4Gaia::organizationalRulesModel::Rule{description};
key metaRE4Gaia::functionalRefinementTree::Function {name};

/* Keys for Gaia metamodel */
key metaGaia::organizationalRulesModel::Rule{description};
key metaGaia::roleModel::Role {name};
key metaGaia::roleModel::Activity {name, role};
key metaGaia::environment::Resource {name};
key metaGaia::environment::Resource {name};
key metaGaia::interactionModel::Protocol {name};
```

5.4.4 Relaciones *top-level* y Relaciones no *top-level*

Una transformación contiene dos tipos de relaciones: *top-level* y no *top-level*. La ejecución de la transformación requiere que se cumplan todas sus relaciones *top-level*, mientras que las relaciones no *top-level* simplemente se requiere que se cumplan cuando se invocan directamente o de forma positiva a través de una cláusula *where* en otra relación. Distinguiremos sintácticamente una relación top-level porque su definición comenzará con la palabra reservada *top*.

Por ejemplo, en la relación *RoleModelMapping* mostrada a continuación, se mapea un Modelo de Roles de RE4Gaia a un Modelo de Role en Gaia.

```
/* Maps a Role Model */
top relation RoleModelMapping{
        checkonly                    domain                    source
domain1:metaRE4Gaia::requirementsRoleModel::RequirementsRoleModel{
          project = p:metaGaia::Project{}
        };
        enforce                     domain                    target
domain2:metaGaia::roleModel::RoleModel{
          system = os:metaGaia::System{}
        };
         when{
          ProjectMapping(p,os);
         }
         where{
          RoleMapping(domain1,domain2);
         }
}
```

Al llamar en la relación *RoleModelMapping*, dentro de su clausula *where* a la relación *RoleMapping*, requiere que se satisfaga la regla *RoleMapping*.

```
    /* Maps a single role */
    relation RoleMapping {

        varName : String;
        varDescription : String;

            checkonly                    domain                    source
domain1:metaRE4Gaia::requirementsRoleModel::RequirementsRoleModel  {
             roles = rrm:metaRE4Gaia::functionalRefinementTree::Role{
                  name = varName,
                  description = varDescription
             }
            };

            enforce                      domain                    target
domain2:metaGaia::roleModel::RoleModel  {
             roles = rm:metaGaia::roleModel::Role{
               name = varName,
               description = varDescription
             }
            };
      }
```

5.4.5 Cláusulas Check y Enforce

Utilizaremos estas cláusulas para definir si se debe forzar o no que el dominio destino cumpla una relación. Si definimos domino con la clausula *checkonly*, simplemente se comprobará si existe una coincidencia válida en el modelo relevante que satisface la relación. Cuando una transformación se ejecuta la dirección de un dominio con la cláusula *enforce*, y la comprobación falla, se modifica el modelo destino para satisfacer la relación.

A continuación se mostrará un ejemplo del uso de estas clausulas:

```
      /* relation metaGaia */
      top relation ProjectMapping {

        varName : String;
        varDescription : String;

            checkonly domain source m1:metaRE4Gaia::Project  {
             name = varName, description = varDescription
            };

            enforce domain target m2:metaGaia::System  {
             name = varName, description = varDescription
            };
      }
```

Si ejecutamos la transformación en la dirección de *source*, entonces para cada Project en el modelo metaRe4Gaia la relación primero comprobará si existe un elemento *System* con el mismo nombre en el modelo metaGaia. Si no existe, se crea un nuevo elemento *System* con el nombre especificado en la variable *varName*.

5.4.6 Ejecución de las reglas

La definición de los meta-modelos origen y destino y sus correspondientes reglas de transformación, nos permiten que podemos ejecutar estas reglas en un motor de transformaciones. En nuestro caso de estudio, expuesto con detalle en el capítulo 7, se definieron los meta-modelos y reglas de transformación mediante la herramienta Medini QVT. Medini QVT es una herramienta basada en el framework de modelado Eclipse, que nos permite construir meta-modelos escritos en el lenguaje Ecore e instanciarlos mediante el lenguaje de intercambio de meta-datos XMI. Las reglas en QVT Relations se definen en ficheros de extensión .qvt y puedes aplicarse sobre los modelos que especifiquemos en la configuración de la herramienta.

La figura 5-16 muestra una captura de la instancia del meta-modelo origen, referente al caso de estudio detallado en el punto 7.2.

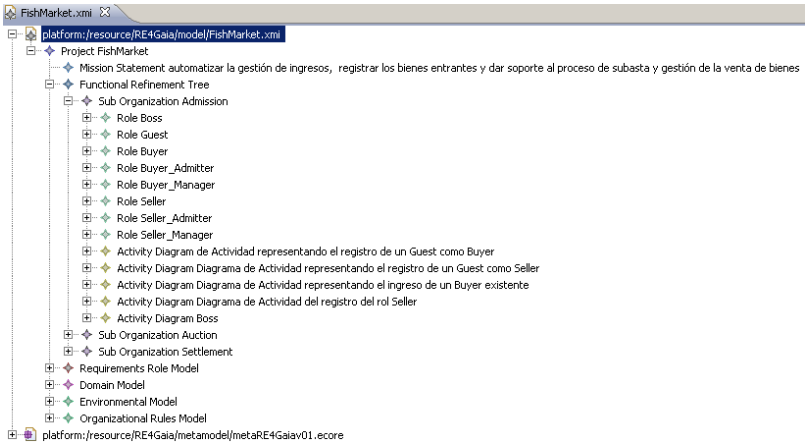


Figura 5-16 Instancia XMI del modelo origen

Tras ejecutar las reglas de transformación, se obtuvo una especificación de análisis en Gaia, codificada en XMI, tal como se muestra en la captura figura 5-17.

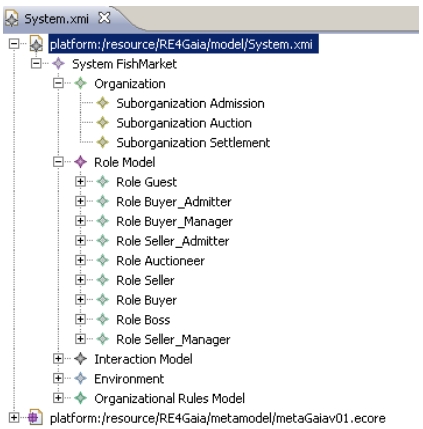


Figura 5-17 Modelo XMI resultante tras aplicar las transformaciones

5.5 Conclusiones

En este capítulo se ha expuesto como se ha dado soporte de este trabajo a los conceptos de Desarrollo de Software Dirigido por Modelos, aplicando los estándares propuestos en la arquitectura MDA por la OMG. Se ha empleado la arquitectura de meta-modelado MOF, utilizándose el lenguaje *Ecore* como lenguaje de definición de lenguajes en el modelo M3. A partir de este lenguaje se definieron los meta-modelos, en el nivel M2, de nuestra propuesta: el meta-modelo RE4Gaia y el meta-modelo de análisis de Gaia. Son éstos los meta-modelos se definirán las reglas que transformación de modelos, expresados en lenguaje *QVT Relations*.

La aplicación de técnicas de Desarrollo de Software Dirigido por Modelos nos permitirá aplicar un enfoque que potenciará la reutilización del software, la mantenibilidad y la adaptación a cambios tecnológicos.

6 Proceso de Desarrollo

En este capítulo se presenta el proceso de desarrollo propuesto, para el modelado de los requisitos para construir un sistema Multi-Agente en Gaia. Esta guía abarca las fases tanto de modelado como de análisis de requisitos del sistema.

6.1 Introducción

Las aplicaciones software son productos complejos muy difíciles de desarrollar y verificar. Frecuentemente el software exhibe comportamientos inesperados que pueden causar problemas y daños. Por esta razón, los investigadores prestan especial atención a entender y mejorar la calidad del software desarrollado. Una de estas direcciones de investigación se basa en el estudio y mejora del proceso con el que se desarrolla el software. Se asume que existe una relación directa entre la calidad del proceso y la calidad del software desarrollado. El área de investigación relacionada con estos aspectos se refiere a estos procesos utilizando el término de *proceso software*.

Un **proceso software** es un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, que son necesarios para concebir, desarrollar, instalar y mantener un producto software (Fuggetta, 2000). Un proceso de desarrollo de software tiene como propósito la producción eficaz y eficiente de un producto software que se adapte a las necesidades del cliente.

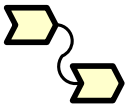
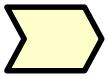
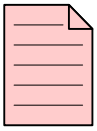
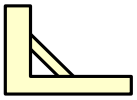
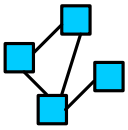
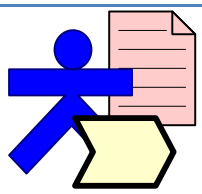
Por otra parte, Sommerville (Sommerville, 2002) define **modelo de proceso de software** como "Una representación simplificada de un proceso de software, representada desde una perspectiva específica. Por su naturaleza los modelos son simplificados, por lo tanto un modelo de procesos del software es una abstracción de un proceso real". Los modelos del proceso de software facilitan la comprensión del proyecto que se va llevar a cabo.

Los modelos de proceso de software describen los procesos de software mediante lenguajes de modelado de procesos. Uno de los más relevantes es el llamada ***Software Process Engineering Metamodel*** o **SPEM** (Object Management Group, 2005) propuesto por la OMG (Object Management Group, 2009). SPEM es un meta-modelo propuesto para utilizar como notación para la definición de procesos de desarrollo de software y sus componentes. Su alcance se limita a los elementos mínimos necesarios para definir dichos procesos sin añadir características específicas de un dominio o disciplina particular; pero sirve para métodos y procesos de diferentes estilos, culturas, niveles de formalismo, o modelos de ciclos de vida. No es un lenguaje de modelado de procesos en general, ya que está orientado a los procesos software. La Tabla 6-1 muestra los elementos principales de SPEM.

SPEM se basa en la idea se basa en la idea de que el proceso de desarrollo de software es una colaboración entre entidades activas abstractas llamadas *roles* de proceso, las cuales llevan a cabo operaciones llamadas *actividades* dentro de entidades tangibles y concretas llamadas *productos de trabajo* (Jacobson & Jacobson, 2005).

Muchos *roles* interactúan o colaboran intercambiando *productos de trabajo* y lanzando la ejecución de ciertas *actividades*. La meta global de un proceso es llevar un conjunto de productos de trabajo a un estado bien definido.

Tabla 6-1 Elementos de definición de procesos SPEM

Elemento	Notación Gráfica	Descripción
Proceso		Representa un proceso completo. Es un conjunto de descripciones de proceso internamente consistente que puede ser reutilizado para definir procesos mayores.
Fase		Representa una fase de un proceso de desarrollo software. Es una especialización de Definición de Trabajo.
Definición de Trabajo		Representa un conjunto de tareas. Es un tipo de operación (o tarea compleja) que describe el trabajo desarrollado en el proceso.
Producto de Trabajo		Es cualquier elemento producido, consumido o modificado por un proceso. Puede ser un fragmento de información, un Documento, un Modelo, código fuente, etc.
Encargado de Proceso		Representa el rol primario que ejecuta y es dueño de una Definición de Trabajo.
Rol de Proceso		Define las responsabilidades sobre un determinado Producto de Trabajo y los roles que ejecutan y ayudan en Actividades específicas.
Actividad		Es la sub-clase principal de Definición de Trabajo. Describe el conjunto de tareas, operaciones y acciones que son ejecutadas por un Rol de Proceso.
Documento		Representa un documento generado en un Proceso.
Guía		Los elementos guía pueden asociarse a cualquier elemento SPEM para proveer información más detallada sobre el elemento asociado. Ejemplos: Guías Técnicas, Métricas, Ejemplos, Perfiles UML, Patrones, etc.
Modelo		Representa los modelos utilizados en el proceso de desarrollo software. Ejemplo: modelo de clases, modelo conceptual, modelo dinámico, modelo de agentes, modelo de interacción, etc.
Paquete de Proceso		Es un contenedor que contiene e importa elementos de descripción de procesos.

6.2 Disciplinas en RE4Gaia

Una disciplina se define como una colección de tareas relacionadas con un "área de interés" en el proyecto global. El particionado de Actividades de esta forma implica que las Guías

asociadas y Productos de Trabajo resultantes están categorizados bajo el mismo tema. La agrupación de tareas en disciplinas ayuda a entender un proyecto desde la visión tradicional de un proyecto en cascada.

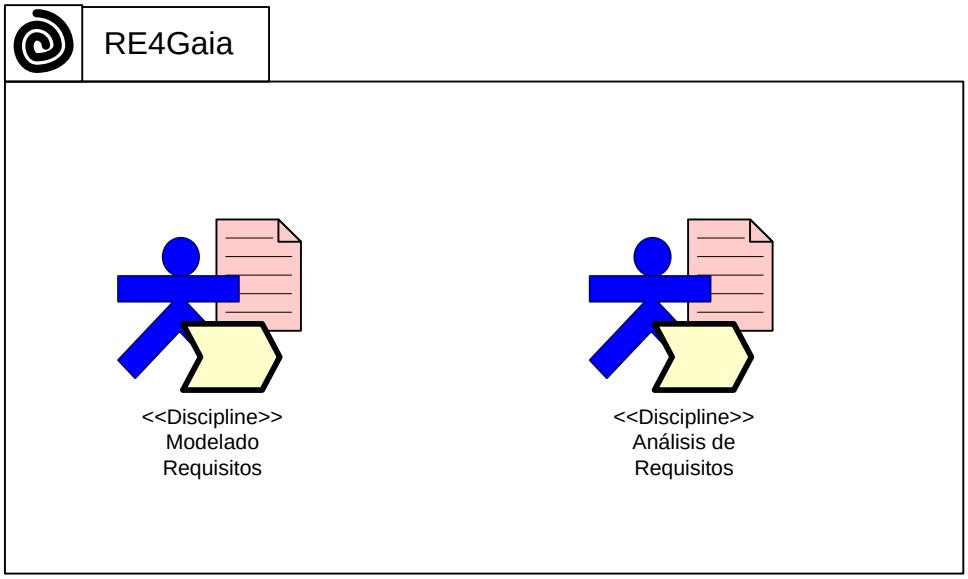


Figura 6-1 Disciplinas del proceso RE4Gaia

En RE4Gaia existen dos disciplinas, tal como se muestra en la figura 6-1:

- **Modelado de Requisitos**: en esta disciplina se incluye el conjunto de tareas necesarias para construir una especificación de requisitos.
- **Análisis de Requisitos**: el objetivo es refinar los artefactos producidos en la disciplina de análisis y acabar construyendo una especificación final de requisitos que servirá de entrada para construir una especificación Gaia.

6.2.1 Modelado de Requisitos

En la disciplina de Modelado de Requisitos (figura 6-2) se implica al rol Analista de Sistema. Este rol será el responsable de realizar las actividades: Definir Misión del Sistema, Construir ARF, Identificar Roles e Identificar Entidades de Dominio.

Respecto a los modelos y documentos, en esta disciplina se incluyen: el documento Misión del Sistema y los modelos Árbol de Refinamiento de Funciones, Modelo de Roles y Modelo de Dominio.

La **misión del sistema** escribe el propósito el sistema en una o dos frases. Su propósito determinado de una forma precisa qué es lo que debe hacer el sistema.

El **Árbol de refinamiento de funciones** se utiliza para descomponer sistema en sub-organizaciones, roles y funciones. En este contexto entenderemos como rol a una entidad abstracta que provee múltiples funciones al sistema.

El **modelo de roles** contiene los roles que participarán sistema y las relaciones de herencia detectadas entre ellos.

Por último en el **modelo dominio**, se definen las entidades identificadas en el dominio problema. Adicionalmente se recogen las relaciones de asociación y herencia entre estas entidades.

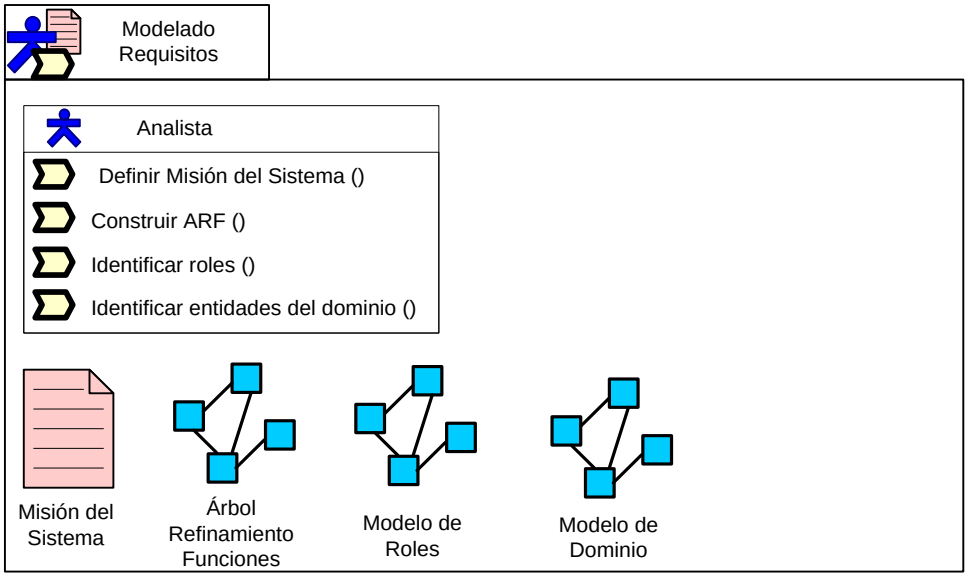


Figura 6-2 Explosionado de la Disciplina Modelado de Requisitos

6.2.2 Análisis de Requisitos

La Disciplina de Análisis de Requisitos (figura 6-3) incluye al rol Analista. Este rol tendrá la responsabilidad de llevar a cabo las tareas de Crear Diagramas de Actividad, Construir Modelo de Entorno e Identificar Reglas Organizacionales.

Además en esta disciplina se incluyen tres *productos de trabajo* (los diagramas de Actividad y Entorno y el documento Reglas Organizacionales).

El **diagrama de actividad** representa el flujo de trabajo, mostrando la secuencia de pasos operacionales de los componentes de un sistema. En este contexto se utilizará para modelar el flujo de trabajo que llevan a uno o varios roles para conseguir sus objetivos. Con la construcción este modelo identificaremos la lista de tareas y protocolos que ejecuta un rol.

En el **modelo entorno** se identificará para cada rol, a que recursos puede acceder y con qué permisos estar autorizado.

El último documento las **reglas organizacionales** se establecen restricciones en el comportamiento global de la organización.

Finalmente, la actividad incluye la Guía llamada "**Guía de Construcción de Diagramas de Actividad**". En el apartado 4.2.2.1 se recogen las guías para construir correctamente los diagramas de actividad en RE4Gaia.

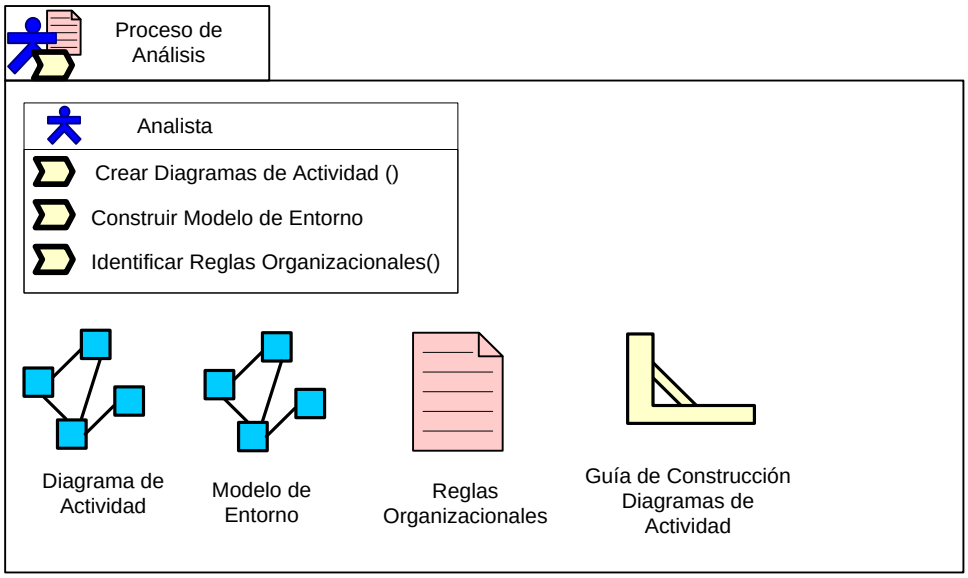


Figura 6-3 Explosionado de la Disciplina Proceso de Análisis

6.3 Descripción del proceso

En esta fase se muestra el proceso para aplicar correctamente RE4Gaia.

6.3.1 Proceso completo

En RE4Gaia se proponen dos fases: *Modelado de Requisitos* y *Análisis de Requisitos*. Cada fase produce un *documento* está compuesto de la suma de todos los modelos y documentos de la definición de trabajo que incluye cada fase. Primero se realizan las actividades de la fase Modelado de Requisitos, produciéndose la *especificación de requisitos*. A continuación se realizan las tareas de la fase de Análisis de Requisitos, produciéndose la *especificación de requisitos refinada*. En este punto puede volver a realizarse la fase de moderada requisitos, en el caso de que se encuentre algún tipo de inconsistencia en la especificación, o puede darse por finalizado el proceso.

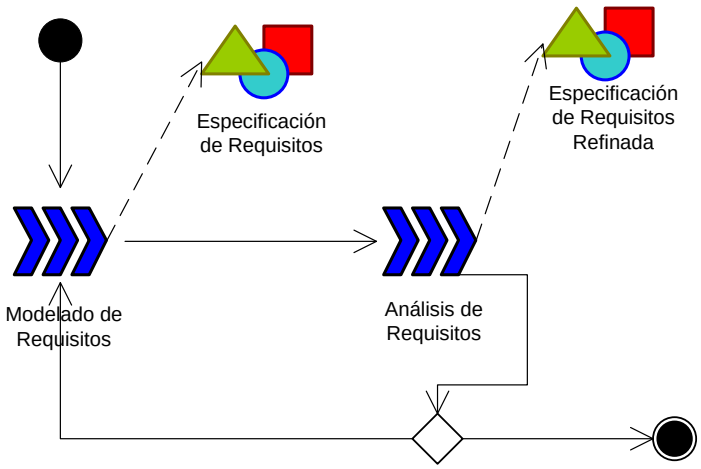


Figura 6-4 Proceso completo en RE4Gaia

6.3.2 Fases / Iteraciones del proceso

A continuación describirnos cada una de las fases del proceso.

6.3.2.1 Fase de Modelado

El objetivo de la fase de análisis es identificar el objetivo global del sistema, identificar las sub-organizaciones, Roles y Entidades que se precisan para construir el sistema. La figura 6-5 muestra el diagrama de proceso correspondiente a esta fase.

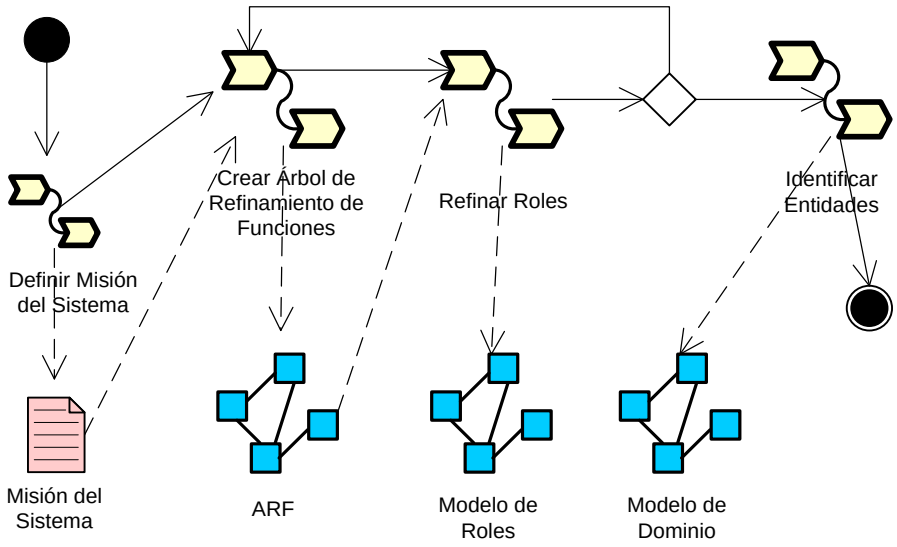


Figura 6-5 Fase de Modelado de Requisitos en RE4Gaia

El primer paso consiste en definir la Misión del Sistema, de esta actividad surgirá el documento llamado Misión del Sistema. Esta visión del sistema define la misión global del sistema y se usará como entrada para la creación del árbol *de refinamiento funciones*. De este modo se descompone la misión en: sub-organizaciones, roles y funciones. Con esta información se construirá el ARF. La lista de roles identificada la tarea anterior, se utilizará como entrada para la actividad *refinar roles*. Aquí se analizarán posibles coincidencias estructurales. Si se considera oportuno puede decirse volver a la fase anterior para actualizar los roles del ARF; o bien, se pone continuar a la fase siguiente. La última fase, *identificar entidades*, se construye el modelo dominio a partir de las entidades identificadas y las relaciones de asociación y herencia entre ellas.

6.3.2.2 Fase de Análisis de Requisitos

El proceso de Análisis de Requisitos implica la creación de dos modelos: Diagramas de Actividad, Modelo de Entorno; y la definición de las Reglas Organizacionales.

La primera actividad esta fase, es la creación de los diagramas de actividad. Se utilizarán como entrada el árbol de refinamiento funciones y el modelo de roles de la fase de modelado requisitos. Como resultado se crearán los diagramas de actividad que estime necesarios. Cuando se finaliza se pasa a la siguiente fase, desarrollo del modelo entorno. Aquí se toman como entrada el modelo de roles y el modelo dominio de la fase de modelado requisitos. El diagrama resultante será modelo entorno. Por último se definirán las reglas organizacionales, tomando como entrada el modelo roles de la fase de modelado requisitos y el modelo entorno. Como resultado se producirá el documento llamado reglas organizacionales.

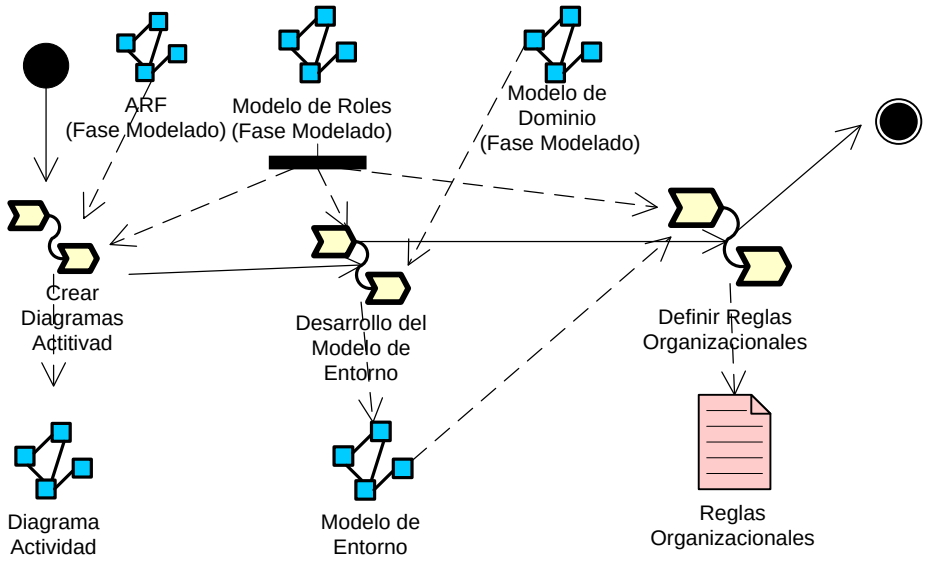


Figura 6-6 Fase de Análisis de Requisitos en RE4Gaia

6.4 Conclusiones

Un proceso software es un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, que son necesarios para concebir, desarrollar, instalar y mantener un producto software (Fuggetta, 2000). La calidad del proceso del software influye directamente en la calidad del producto software generado. Por ello es necesario utilizar técnicas de modelo del proceso de software, para definir de una forma precisa estos procesos. Existen múltiples lenguajes propuestos para modelar los procesos de software. En la definición del proceso desarrollo de esta propuesta se utilizó SPEM (Object Management Group, 2005). El uso de SPEM proporciona la ventaja de haber sido propuesto OMG y poderse integrar fácilmente con el resto de sus estándares, como por ejemplo el lenguaje de modelado UML. Es posible modelar procesos con SPEM, gracias al uso de perfiles de UML, lo que permite disfrutar del gran apoyo mediante herramientas a este lenguaje de modelado.

El modelo de proceso de RE4Gaia está compuesto de dos fases: modelado y análisis:

- **Modelado**. El objetivo de la fase de análisis es identificar el objetivo global del sistema, identificar las sub-organizaciones, definir los Roles y Entidades que se precisan para construir el sistema. En esta fase se parte de la Definición del Sistema, la cual se usa como punto de partida para la creación del ARF. A partir de este se crea el Modelo Roles, pudiéndose identificar relaciones de herencia entre ellos o posibles incompatibilidades. En este último caso, debería volverse a refinar el ARF agregando o eliminando roles. Por último se crean modelo de dominio, que incluye las entidades relevantes de este y las relaciones de asociación y herencia entre ellas.
- **Análisis**. El proceso de Análisis de Requisitos implica la creación de dos modelos: Diagramas de Actividad, Modelo de Entorno; y la definición de las Reglas Organizacionales. Primero se crean los diagramas de actividad, a partir del modelo roles y el ARF. En el siguiente paso se desarrolló el modelo entorno a partir de los modelos de roles y de dominio. Por último se definen las reglas organizacionales.

7 Caso de Estudio

En esta sección se describirá el caso de estudio utilizado para validar la propuesta. Se incluirán los modelos generados tras la aplicación de RE4Gaia y los modelos de la fase de Análisis de Gaia creados a partir de esta información.

7.1 Introducción

Este caso de estudio está basado un mercado de venta de pescado mediante un sistema de subasta. Esta propuesta se utilizó en los trabajos de (Napoli, Giordano, & Furnari, 1996), (Rodríguez, Noriega, Sierra, & Padget, 1997), (Instituto de Investigación en Inteligencia Artificial (IIIA), 1998).

El primer rol que participa en el sistema es el jefe ***(Boss)***. Es el encargado de abrir el mercado. Además tiene la función de actuar de supervisor del proceso de subasta, asumiendo la máxima autoridad en la casa de subastas.

Por otra parte, el proceso de subasta se dividirá en tres fases: ingreso, subasta y venta.

La primera fase es la de **ingreso**. Si un **comprador (*Buyer*)** quiere entrar en el mercado, primero tiene que registrarse en la oficina de registro. Si se acepta su solicitud de ingreso, entonces puede entrar en la sala de subastas. Los compradores *nuevos* además deben abrir una cuenta de crédito para efectuar los pagos en el mercado. Lo usuarios ya *existentes* deben validar su crédito disponible.

Mientras los **vendedores** efectúan el ingreso, los vendedores llevan los bienes a subastar a la oficina de ingreso. Las cajas de pescado que traen los vendedores, se llevan del barco a la oficina de ingreso. Las cajas de pescado se descargan de los barcos a la mañana en el puerto y se transportan al mercado.

El ingreso de vendedores lo controla el ***Seller Admitter***. El empleado *Seller Admitter* fija para cada caja su peso, calidad del pescado y precio. Eventualmente manda la lista de bienes al *Auctioneer*. Mientras el proceso de subasta se lleva a cabo, el vendedor espera hasta que sus bienes se venden. Solo cuando esto ocurre, el vendedor recibe el dinero del *Seller manager*.

La siguiente fase es la ronda de subastas, que tiene lugar en la sala de subastas. Esta es la fase más importante del sistema. Los compradores pujan por los bienes ofrecidos por el *subastador*. Este anunciará los precios en orden descendente siguiente el protocolo *Downaward Bidding Protocol*.

Una ronda puede iniciarse o reiniciarse cuando el Jefe se lo autoriza al Subastador. El Jefe puede decidir que una ronda no puede iniciarse si:

- No existen al menos tres compradores.
- No existen suficientes bienes para subastar todavía.

Cuando el Jefe da el visto bueno, el Subastador comienza la ronda de subasta. Primeramente, el Subastador envía a cada comprador la lista de compradores que participaran en la ronda, los bienes a subastar y el bien que se subastará. Después de esto, el Subastador comienza a anunciar los precios a todos los compradores en la sala de subastas. Cuando un bien se vende, entonces se

anunciar al vendedor que hay una nueva venta. A continuación se actualizan las ganancias del Vendedor.

Existen múltiples situaciones que pueden darse en una ronda de subasta y deben controlarse:

- **Saldo Correcto**. El comprador puja por un bien y gana la ronda. Si tiene suficiente saldo, entonces puede comprar el bien subastado.
- **Puja no soportada**. Un comprador puja por un bien pero no tiene saldo suficiente para comprarlo. En este caso el Subastador reinicia la ronda, incrementando el precio en un 25%.
- **Colisión**. Si dos o más compradores pujan al mismo tiempo el Subastador declarará una colisión. En ese caso se reinicia la ronda, incrementándose el precio inicial en un 25%.

La última fase es la de **Ventas**. En este parte del sistema, los compradores se personan en la oficina de ventas antes de salir del mercado, con el objetivo de recoger los extractos que describen sus compras y pagarlas. Mientras tantos, los vendedores recogen sus ganancias una vez que se han vendido sus bienes. Los compradores además pueden ir a la oficina de ventas a actualizar su crédito o llevar a cabo otras operaciones.

El resto del capítulo se estructurará de la siguiente manera: en la sección 7.2 se mostrará la aplicación de la fase de requisitos de RE4Gaia, en la sección 7.2.2 se mostrará el análisis de requisitos, la sección 7.3 muestra la fase del análisis en Gaia generada a partir de los artefactos generados en las fases anteriores. Finalmente la sección muestra las conclusiones de este capítulo.

7.2 RE4Gaia

A continuación de mostrará la resolución del caso de estudio aplicando RE4Gaia.

7.2.1 Modelado Requisitos

El objetivo de la fase de Modelado de Requisitos (MR) es adquirir y representar los requisitos del software. Esta fase comprende con la definición de la Misión del Sistema, y la creación del ARF, Modelo de Roles y el Modelo de Dominio. La Misión del Sistema establece los objetivos principales de la organización. El ARF ayuda a determinar que sub-organizaciones componen la organización y los roles que participan en ella. Con el Modelo de Roles de Requisitos, se pretende detectar relaciones de herencia entre roles y razonar sobre sus relaciones estructurales, detectando posibles inconsistencias. Finalmente en el Modelo de Dominito se detectar las entidades que podrían ser recursos del sistema.

7.2.1.1 Misión del Sistema

El Misión del Sistema representa el principal objetivo del sistema. Está escrito en lenguaje natural en uno dos párrafos típicamente. El objetivo es determinar de una forma clara y concisa, qué es lo que el sistema debe hacer.

Respecto al caso de estudio, la misión del Mercado de venta de Pescado es "*automatizar la gestión de ingresos, registrar los bienes entrantes y dar soporte al proceso de subasta y gestión de la venta de bienes*". La actividad principal, la subasta, comprende a otras sub-tareas como la gestión de conflictos: el control de que existen suficientes compradores, gestión de colisiones en las pujas y gestión de intentos de compra sin suficiente saldo.

Tabla 7-1 ARF del sistema *Fish Market* en formato tabular

Sub-organización	Rol	Función
Admission	Guest	• Request register • Request open credit account • Exit
	Buyer	• Change scene • Request Credit Update • Exit
	Buyer admitter	• Process Create Account request. • Process Register request. • Request Open Account
	Buyer manager	• Update credit • Update buyer List. • Process Open Account request • Process Exit request
	Boss	• Start the system
	Seller	• Submit good • Exit
	Seller admitter	• Process Registration request. • Process New Account request • Process Good Submissions • Fix Good Price • Send good list • Notify New Account
	Seller manager	• Process Exit request • Process New Account Notify • Process New Good Submission • Update Seller List
Auction	Auctioneer	• Request the Good List • Request the Buyer List • Init Auction • Ask for New Round • Send Current Price • Read Bid offers • Count Offers • Decrease Good Price • Increase Good Price • Declare Winner • Communicate final sale
	Buyer	• Receives Current Round Price • Bid for a Good • Receives the final sale
	Buyer Manager	• Process a Buyer List Request
	Boss	• Give permission to Start round • Close Auction
	Seller	• Read Auction Information
	Seller Admitter	• Request Good List
	Seller Manager	• Process Good List Request
Settlement	Seller	• Read purchase information • Receives a Sale Notification • Change scene • Exit
	Seller Manager	• Update Seller List • Exit
	Auctioneer	• Process a bad credit notification • Sanction Buyer
	Buyer Manager	• Check Credit • Process Purchase • Notify Bad Credit • Sanction Buyer • Update Credit

7.2.1.2 Árbol de Refinamiento de Funciones

El ARF descompone el sistema en funciones que toman como nodo raíz la misión del sistema. A partir de la misión del sistema, obtenemos los nodos intermedios, que representan a las sub-organizaciones en el nivel 1 y a los roles en el nivel 2. Finalmente los nodos hoja representan las funciones. Una función puede llevarse a cabo por un rol, en el contexto de una sub-organización, con independencia de si el rol necesita colaborar con otros roles o no.

Respecto a nuestro caso de estudio, la Tabla 7-1 muestra el ARF en un formato tabular. Se muestra como tabla debido a limitaciones de espacio.

7.2.1.3 Modelo de Roles de Requisitos

El Modelo de Roles de la fase de requisitos muestra los roles que previamente se identificaron en el ARF y permite especificar relaciones de herencia entre ellos. En nuestro caso de estudio se identificaron los roles: *Guest*, *Buyer*, *Seller*, *Boss*, *Buyer Manager*, *Auctioneer*, *Seller Admitter*, *Buyer Admitter* and *Seller Manager*. Respecto a las relaciones de herencia, se identificaron herencias desde el rol *Guest* a los roles *Buyer* y *Seller*, representando el hecho que todo lo que pueda hacer el rol *Guest*, lo pueden hacer los roles *Buyer* y *Seller*. Esta información se muestra en la figura 7-1.

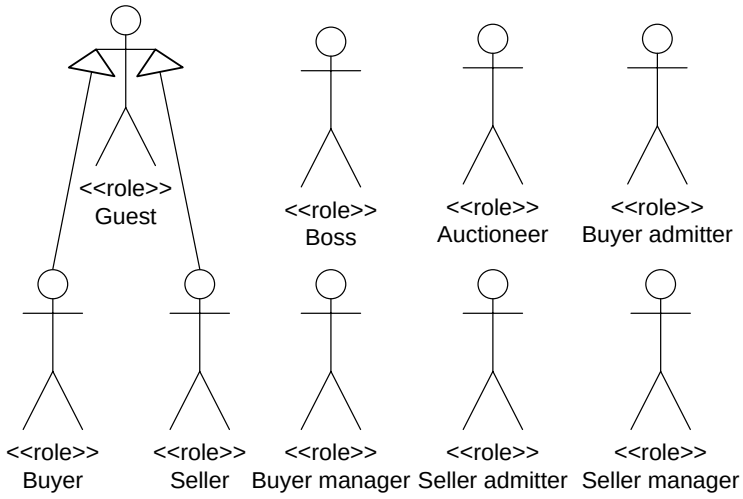


Figura 7-1 Modelo de Roles de Requisitos

7.2.1.4 Modelo de Dominio

Las entidades principales del dominio y sus relaciones que se encontraron en el caso de estudio se representaron en la figura 7-2. Se ha identificado que la entidad *Person* comparte propiedades comunes con las entidades *Seller* y *Buyer*. Un *Buyer List* se relaciona con múltiples *Buyer* y un *Good List* se relaciona con múltiples *Good*. Un *Good* pertenece a un *Seller*, y un *Bidding Round* está compuesto por múltiples *Bid*. Cada *Bid* tiene un *Buyer List*, con múltiples *Price* y *Good List* asociados.

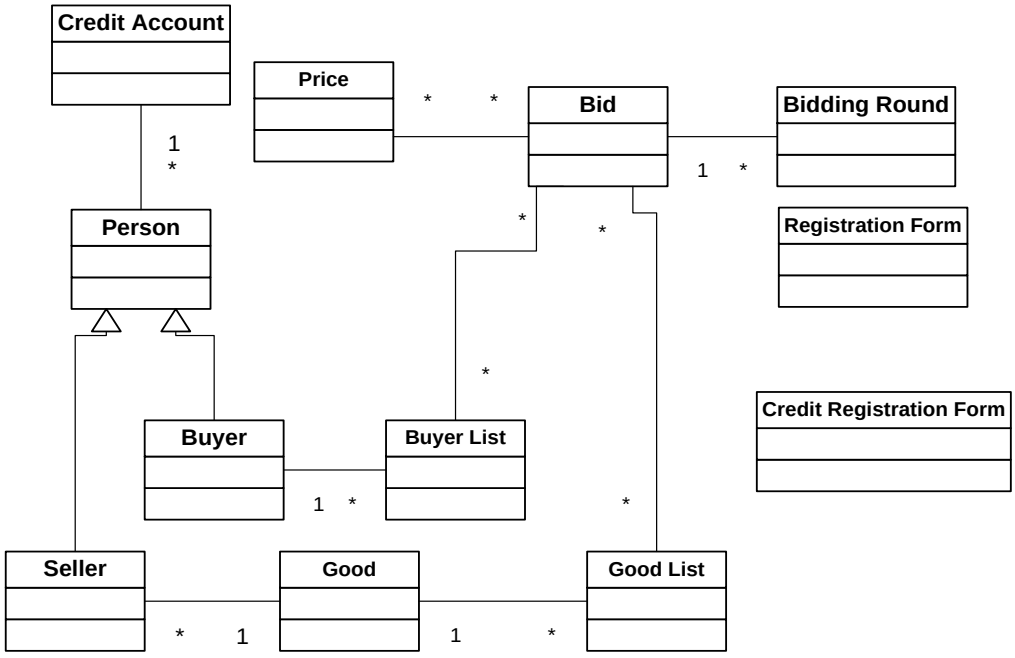


Figura 7-2 Modelo de Dominio

7.2.2 Análisis de Requisitos

En esta fase se incluye el análisis de las funciones previamente identificadas en el ARF, usando la información del Modelo de Roles y el Modelo de Dominio. El Diagrama de Actividad se propone para refinar las funciones en: *tareas*, las cuales se llevan a cabo por un único rol; o *protocolos*, las cuales son funciones llevabas a cabo por uno o más roles cooperando. Con las roles identificados en el Modelo de Roles, las entidades del Modelo de Dominio y la información extraída del Diagrama de Actividad, el paso siguiente es identificar los Recursos y Permisos. Finalmente, se establecen el conjunto de reglas organizacionales para regular el comportamiento de la organización definido.

7.2.2.1 Diagrama de Actividad

El Diagrama de Actividad específica las construcciones dinámicas de una organización usando actividades y acciones. Este diagrama muestra una secuencia de pasos mostrando el flujo de datos necesario para realizar las funciones identificadas.

Utilizaremos los Diagramas de Actividad para analizar en detalle cada sub-organización identificada en el ARF. El objetivo de este diagrama es identificar dependencias de roles y refinar las funciones identificadas en tareas y protocolos.

A continuación se mostraran los Diagramas de Actividad necesarios para nuestro caso de estudio. Se comenzará con los diagramas de la **sub-organización de Registro (*Admission*)**, concretamente se construirán cinco. Se emplearán dos diagramas para modelar el proceso de registro de un *Guest* en un *Buyer* o en un *Seller* respectivamente. Los otros diagramas representan el proceso de ingreso de *Buyers* y *Sellers* ya existentes. El último mostrará el comportamiento del rol *Boss* en esta fase.

En la figura 7-3 se representa el proceso de registro de un *Guest* como *Buyer*. Cuando un *Guest* inicia su flujo de ejecución y puede elegir o bien iniciar el protocolo de *Register* o bien salir del sistema mediante la tarea *Exit*. Si se ha optado por iniciar el protocolo *Register*, el siguiente paso es iniciar el protocolo *Open Account* para llevar a cabo la apertura de una cuenta de crédito. Por otra parte, el *Buyer Manager* espera a recibir la solicitud de un protocolo *Register* y a continuación del protocolo *Open Account*, cuando se han llevado a cabo los dos protocolos, iniciar el protocolo *New Account* para indicarle al *Buyer Manager* que ha de realizar la tarea *Update Buyer List*, para modificar la lista de compradores activos.

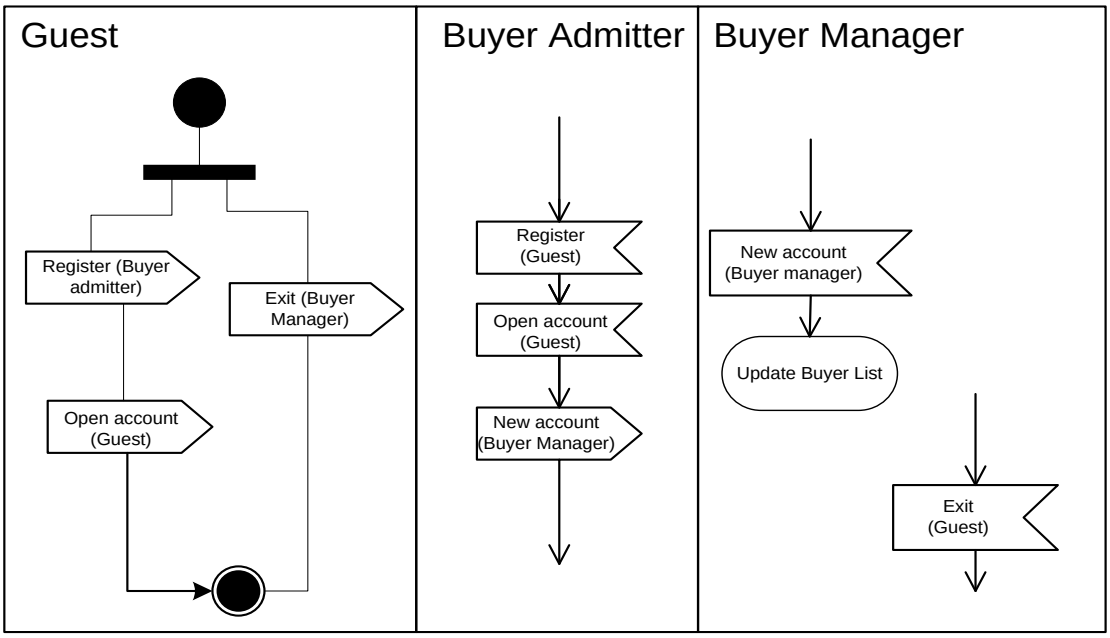


Figura 7-3 Diagrama de Actividad representando el registro de un *Guest* como *Buyer*

El diagrama de la figura 7-4 representa el proceso de registro de un *Guest* como *Seller*. Cuando inicia el flujo de acción el *Guest* puede o bien iniciar el protocolo *Register* o bien salir del sistema ejecutando la tarea *Exit*. El *Seller Admitter* espera peticiones del protocolo *Register* y cuando este se finaliza, llama al protocolo *New Account* para indicarle al Seller Manager que tiene que ejecutar la tarea *Update Seller List* para actualizar la lista de vendedores activos.

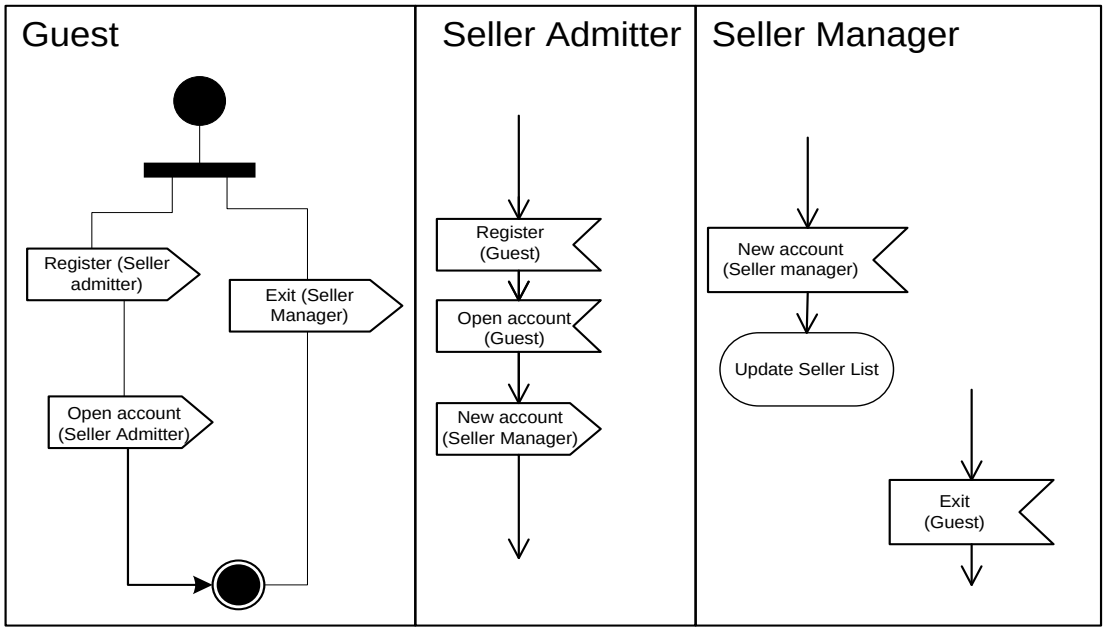


Figura 7-4 Diagrama de Actividad representando el registro de un *Guest* como *Seller*

En el siguiente diagrama (figura 7-5) se muestra el proceso de ingreso de un *Buyer* en el sistema. El *Buyer* debe de iniciar siempre el protocolo *Credit Update* para o bien validar o actualizar su crédito. A continuación el *Buyer* puede elegir cambiar de escena con *Change Scene* o bien salir del sistema iniciando el protocolo *Exit*.

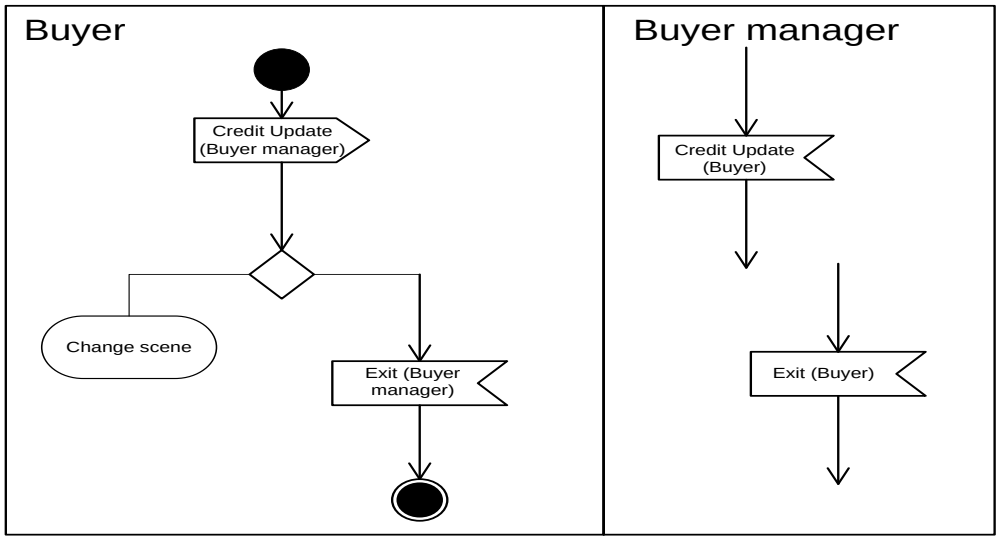


Figura 7-5 Diagrama de Actividad representando el ingreso de un *Buyer* existente

El siguiente diagrama de esta fase (figura 7-6) muestra el proceso de entrada de un *Seller* en el sistema. Un *Seller* puede escoger o bien registrar un bien mediante el protocolo *Submit Good* o bien salir del sistema. Si escogió enviar un bien a subastar, cuando acaba el protocolo *Submit Good* se ejecuta la acción *Change Scene*, de esta forma el *Seller* puede ir a la escena *Auction* para ver las subastas en curso o a la escena *Settlement* para comprobar sus ganancias.

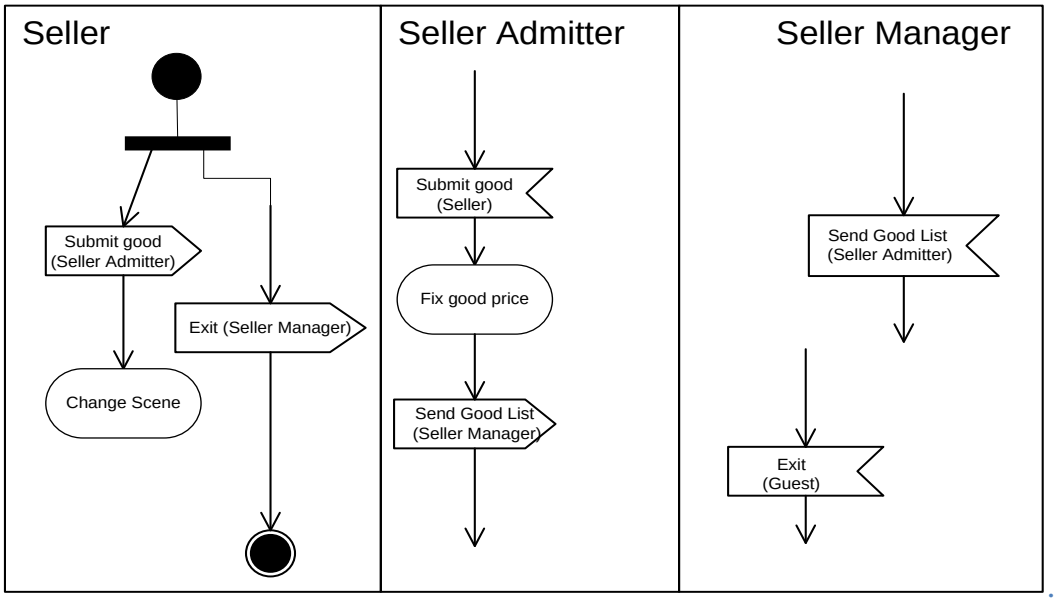


Figura 7-6 Diagrama de Actividad del registro del rol Seller

El último diagrama de esta sub-organización es el de la figura 7-7, el cual muestra el flujo del rol *Boss*. Este pude invocar la tarea *Start the system* para activar el sistema *Fish Market*.

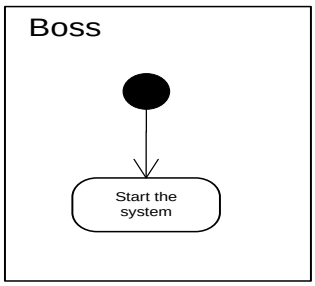


Figura 7-7 Diagrama de Actividad del Rol Boss

A continuación de mostrarán los diagramas creado para representar el flujo de acciones en la **fase de *Auction***. El primero de ellos representa el flujo de acción para el rol *Auctioneer* (Figura 7-8). De acuerdo al diagrama, el *Aucioneer* primer activa los protocolos *Request good listy* y *Request buyer list* en paralelo. Una vez obtiene el resultado de estos dos protocolos, realiza la acción *Init Auction* para preparar una nueva subasta. Cuando finaliza lanza el protocolo *New Round* para pedir autorización al jefe para iniciar la nueva ronda. Si el jefe finalmente da permiso, el *Auctioneer* trasmite el precio actual de la puja mediante el protocolo *Current Price*, a continuación esperar pujas, las cuales trasmiten los *Buyers* mediante el protocolo *Bid*. El *Auctioneer* a continuación cuenta las pujas que recibe mediante la acción *Count Offers*. Si no hay pujas, se decrementa el precio mediante la tarea *Decrease Price* y se vuelve a trasmitir el nuevo precio. Si existen pujas, se comprueba que no hay dos pujas por el mismo precio (colisión). En caso de haber colisión y incrementa el precio y se trasmite a todos los Buyers. Si no hay colisión entonces se declara a un ganador de la ronda. El resultado se comunica mediante el protocolo *Communicate Sale* tanto al Buyer como al Seller Manager.

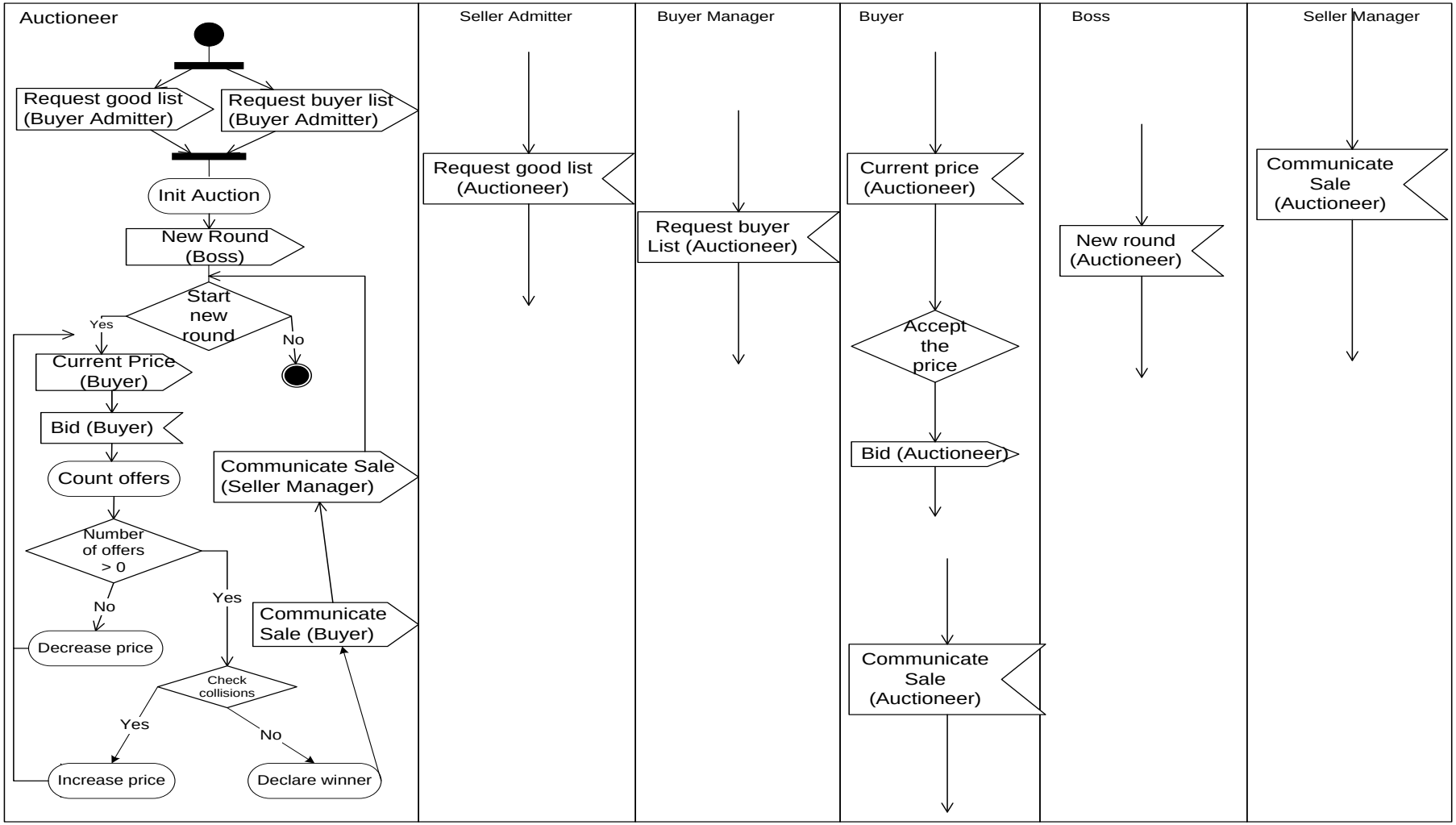


Figura 7-8 Diagrama de Actividad del rol *Auctioneer* en la sub-organización *Auction*

El siguiente diagrama muestra el funcionamiento del rol *Boss* (figura 7-9). Este diagrama modela el comportamiento de que el *Boss* puede parar la subasta en cualquier momento que lo desee llamando a la tarea *Close Auction*. Adicionalmente, el rol *Boss* también puede responder a un protocolo New Round tal como muestra la figura 7-8.

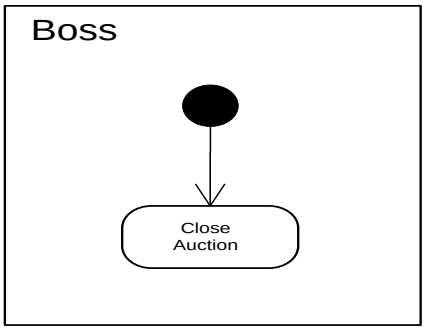


Figura 7-9 Diagrama de Actividad para el rol *Boss* en la sub-organización Auction

El siguiente diagrama se muestra el comportamiento del rol *Seller* (figura 7-10). El *Seller* puede consultar la información de la subasta en cualquier momento mediante *Read Auction Information* y puede decidir tras comprobarla si continuar observando o cambiar de escena.

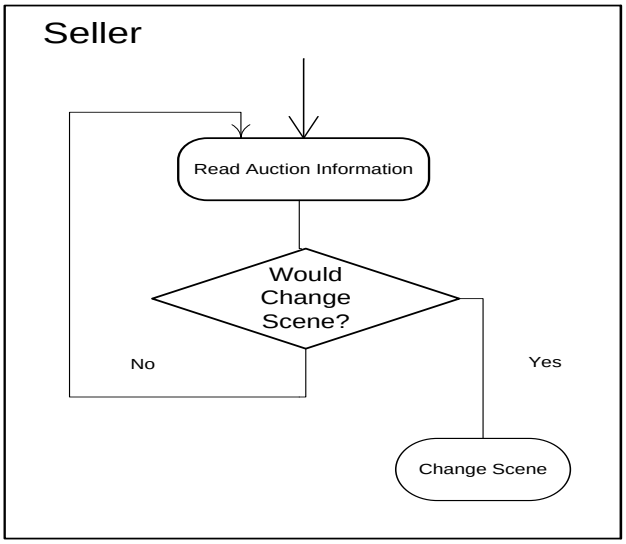


Figura 7-10 Diagrama de Actividada para el rol Seller en la sub-organización Acution

Respecto a la última fase, *Settlement*, construiremos dos Diagramas de Actividad. Cada uno de estos diagramas representa a un rol con comportamiento proactivo (un rol que inicia protocolos), concretamente los roles: *Buyer* y *Seller*. La figura 7-11 muestra el Diagrama de Actividad para el rol *Buyer*. El *Buyer* cuando entra en el escenario puede iniciar el protocolo *Purchase* para interactuar con el *Buyer Manager*. Además se ha añadido una restricción mediante una nota para indicar que el rol *Buyer* debe ser el ganador de una puja para poder iniciar el protocolo. Cuando el rol *Buyer* finaliza el protocolo *Purchase*, pues escoger salir del sistema o cambiar de escena. En la otra partición, el *Buyer Manager* espera respuestas al protocol *Purchase*. El rol comprueba si el rol *Buyer* tiene suficiente crédito. En caso afirmativo entonces se ejecuta al tarea *Update Credit*, en caso contrario se inicia el protocolo *Bad Credit* para comunicar una "compra incorrecta" al rol *Auctioneer*.

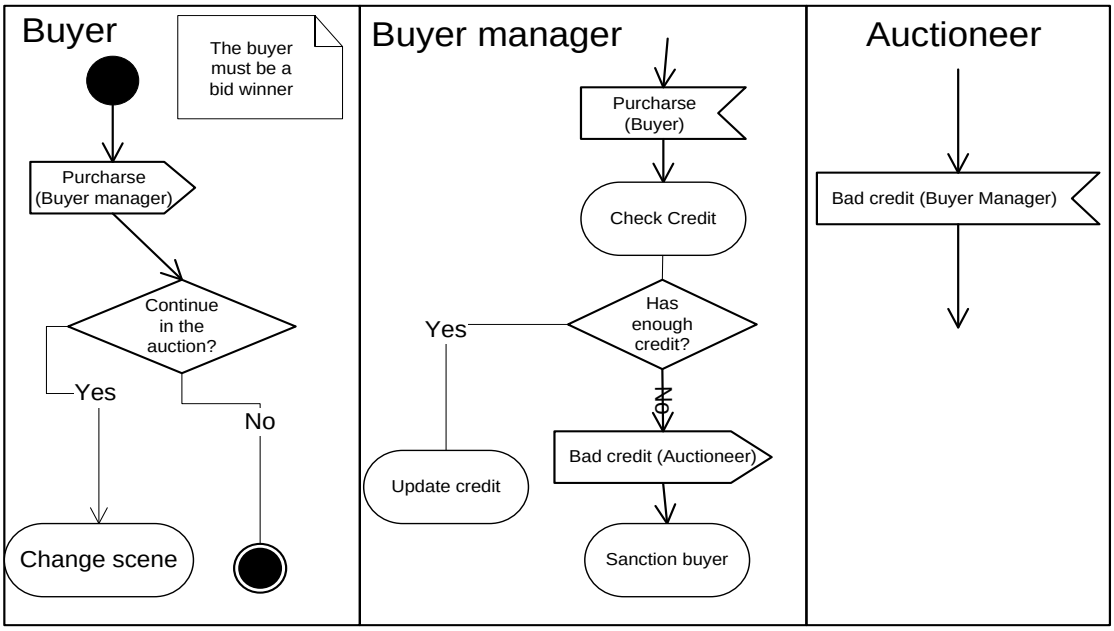


Figura 7-11 Diagrama de Actividad del rol *Buyer* en la sub-organización *Settlement*

El otro diagrama construido, representa el comportamiento del rol *Seller*, tal como se ve en la figura 7-12. El *Seller* cuando entra en la escena y espera que le lleguen notificaciones de ventas (protocolo *SaleRecord*). Una vez la he llegado una puede elegir continuar (espera la venta de más productos) o puede decidir no continuar. En caso de no continuar en la escena, puede elegir cambiar de escena (*Change_Scene*) o salir del sistema (Llama al protocolo *Exit*).

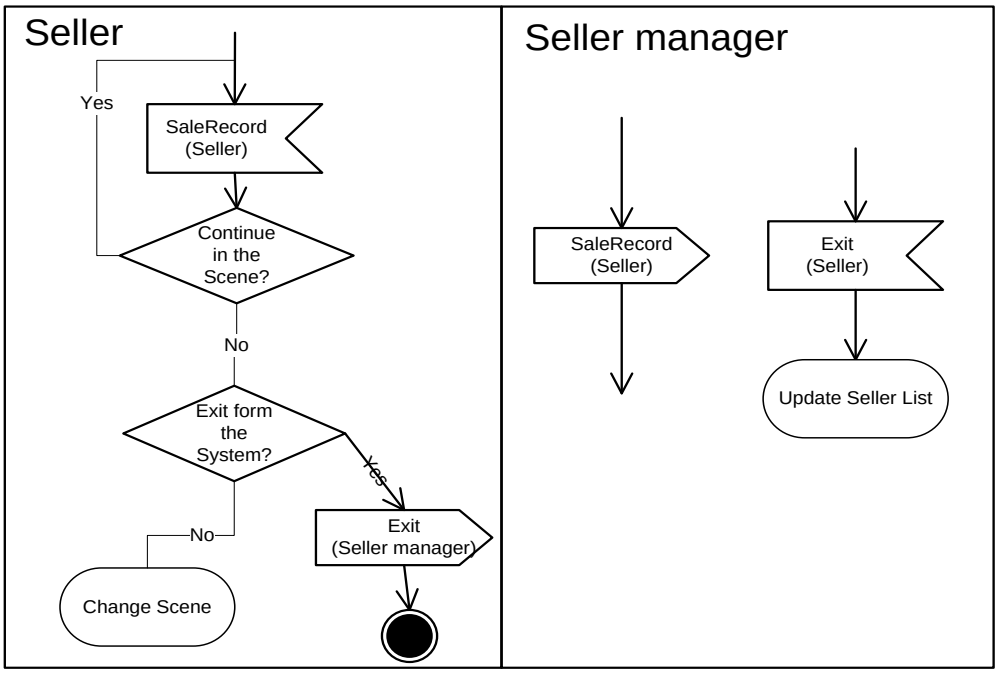


Figura 7-12 Diagrama de Actividad del rol *Seller* en la sub-organización *Settlement*

7.2.2.2 Modelo de Entorno

El Modelo de Entorno es un conjunto de tuplas con la siguiente estructura: role, recurso y permiso. Para cada rol identificado en el ARF, y también especificado en el Modelo de Roles para considerar relaciones de herencia, establecemos los recursos y su respecto permiso concedido al rol. La Tabla 7-2 muestra el Modelo de Entorno construido para el *Fish Market*.

Tabla 7-2 Modelo de Entorno del Fish Market

Rol	Recurso	Permiso
Auctioneer	Bid	Modificar
	Bidding Round	Modificar
	Price	Modificar
Buyer	Buyer List	Leer
	Bid	Leer
	Good	Leer
Buyer Admitter	Buyer	Modificar
	Registration Form	Leer
Buyer Manager	Buyer List	Modificar
	Credit Account	Modificar
Boss	Buyer List	Modificar
Guest	Credit Registration Form	Modificar
	Registration Form	Modificar
Seller	Good List	Leer
	Good	Leer
	Purchase	Leer
Seller Admitter	Good	Modificar
	Good List	Modificar
Seller Manager	Good List	Leer
	Purchase	Leer
	Seller List	Modificar

7.2.2.3 Reglas Organizacionales

El último paso de la fase de Análisis de Requisitos es definir las Reglas Organizacionales. Estas reglas se escriben en lenguaje natural y tienen el propósito de identificar y representar restricciones generales sobre el comportamiento de la organización. Estas reglas pueden ser vistas como responsabilidades de la organización vista como un todo y puede ser de dos tipos:

i. **Propiedades dinámicas de la organización**. Estas definen como debe de comportarse la organización.
ii. **Restricciones de la Organización**. Definen restricciones que la organización debe respetar en cada instante de tiempo.

Respecto al caso de estudio: se identificaron las siguientes reglas organizacionales:

- Un *Buyer* no se puede entrar directamente en la sub-organización *Auction*.

- El *Buyer* debe de registrarse en la sub-organización de Registro antes de entrar en el sistema.
- Un *Buyer* no puede entrar en la sub-organización *Settelment* si no ha ganado una ronda.
- El *Auctioneer* debería de esperar si no hay suficientes bienes a subastar.
- El *Auctioneer* debe de esperar si no hay suficientes *Buyers* en la subasta.

7.3 Análisis en Gaia

En este apartado se mostrarán los artefactos del análisis de Gaia, generados automáticamente a partir de los obtenidos en RE4Gaia.

7.3.1 División del Sistema en Sub-organizaciones

En este paso, se debe identificar las sub-organizaciones que constituyen la organización global. Para nuestro caso de estudio se identificaron tres:

- **Sub-organización *Admission***. Esta sub-organización incluye el registro de los *Guest* en el sistema y la creación de sus cuentas de crédito.
- **Sub-organización *Auction***. En esta sub-organización se realiza el proceso de subasta del pescado. Además en esta sub-organización el *Seller* podrá consultar el estado del proceso de subasta en cualquier momento.
- **Sub-organización *Settlement***. En esta sub-organización los *Buyer* efectúan el pago de los bienes. Durante este proceso, el *Seller* puede esperar la notificación de una venta de un bien suyo.

7.3.2 Modelo de Entorno

El modelo de entorno se compone de los recursos del sistema identificados. En nuestro caso estudio identificamos los siguientes recursos:

- **Bid.** Información de una puja, comprende precio, artículo a comprar y *Buyer*.
- **Bidding Round**. Información de una ronda de subasta, comprende: bien o bienes a subastar y lista de *Buyers.*
- **Buyer Information.** Información personal de un comprador**.**
- **Buyer List.** Lista de compradores activos del sistema.
- **Credit Account.** Información o la cuenta de crédito del mercado.
- **Credit Registration Form.** Formulario para crear una cuenta de crédito en el sistema.
- **Good.** Información o bien a subastar. Esta información incluye: peso, calidad del pescado, precio.
- **Good list.** Lista de bienes a subastar.
- **Price.** Precio actual de un bien a subastar.
- **Purchase.** Información de una venta.
- **Purchase list.** Lista de ventas hechas.
- **Registration Form.** Formulario de registro para entrar en el sistema.
- **Seller List.** Lista de vendedores activos del sistema**.**

7.3.3 Modelo Preliminar de Roles

En este modelo se captura una versión previa de los roles de la organización. Esto roles identifican con independencia de la estructura organizacional. Para representar a los roles, utilizaremos la plantilla presentada en la

Tabla 7-3.

Tabla 7-3 Plantilla de Roles

Rol	*Nombre del rol*
Descripción	*Descripción textual del rol y su comportamiento*
Protocolos y Actividades	*Lista de protocolos actividades que ejecutaron*
Permisos	*Lista de permisos sobre los recursos del entorno*

En nuestro caso estudio identificamos los siguientes roles:

Tabla 7-4 Rol *Auctioneer*

Rol	Auctioneer
Descripción	Rol encargado de gestionar las subastas
Protocolos y Actividades	Request Good List, Request Buyer List, New Round, Current Price, Communicate Sale, Bid, Bad Credit, Init Auction, Count Offers, Decrease Price, Increase Price, Declare Winner.
Permisos	**Modificar**: Bid **Modificar**: Bidding Round **Modificar**: Price

Tabla 7-5 Rol *Boss*

Rol	Boss
Descripción	El Boss es la máxima autoridad del Sistema. Es el encargado de iniciar y detener el sistema cuando lo considere necesario. Además tiene que dar permiso para que se pueda iniciar una subasta.
Protocolos y	New Round, Start System, Close Auction

Actividades	
Permisos	**Modifica:** Buyer List

Tabla 7-6 Rol *Guest*

Rol	Guest
Descripción	Representa a un usuario que está fuera del sistema. Un Guest puede registrarse como Seller o Buyer.
Protocolos y Actividades	Register, Exit, Open Account
Permisos	**Modifica:** Registration form. **Modifica:** Credit Registration form.

Tabla 7-7 Rol *Buyer*

Rol	Buyer
Descripción	Rol que tiene como objetivo realizar pujas por productos en la subasta. Adicionalmente puede actualizar y validar su crédito.
Protocolos y Actividades	Change Scene, Credit Update, Exit, Bid, Purchase, Current Price, Communicate Sale.
Permisos	**Leer:** Good **Leer:** Bid **Leer:** Buyer List

Tabla 7-8 Rol *Buyer Admitter*

Rol	Buyer Admitter
Descripción	Procesa las solicitudes de registro de un *Guest* para convertirse en *Buyer*.
Protocolos y	New Account, Register, Open Account

Actividades	
Permisos	**Modifica:** Registration form. **Modifica:** Account Registration form.

Tabla 7-9 Rol *Buyer Manager*

Rol	Buyer Manager
Descripción	Controla a los *Buyer* que están participando en el sistema. Adicionalmente, actualiza la lista de *Buyers* activos en el sistema.
Protocolos y Actividades	Credit Update, Exit, New Account, Request Buyer List, Purchase, Update Buyer List, Update Credit List, Sanction Buyer, Check Credit.
Permisos	**Modifica:** Credit Account. **Modifica:** Buyer List.

Tabla 7-10 Rol *Seller*

Rol	Seller
Descripción	Rol encargado de registrar productos a subastar en el sistema. Cuando se vende su producto recoge las ganancias.
Protocolos y Actividades	Submit Good, Exit, Sale Record, Change Scene
Permisos	**Leer**: Good **Leer**: Good List **Leer**: Purchase

Tabla 7-11 Rol *Seller Admitter*

Rol	Seller Admitter
Descripción	Rol encargado del registro de Seller y de loss productos que pretende subastar
Protocolos y Actividades	New Account, Register, Open Account, Submit Good, Request Good List, Fix Good Price.

Permisos	**Modificar**: Good **Modificar**: Good List

Tabla 7-12 Rol *Seller Manager*

Rol	Seller Manager
Descripción	Rol encargado de la gestión de los *Sellers* activos en el sistema y del reparto de ganancias.
Protocolos y Actividades	Sale Record, New Account, Exit, Update Seller List
Permisos	**Leer**: Purchase **Modificar**: Seller List **Leer**: Good List

7.3.4 Modelo Preliminar de Interacciones

En esta sección se detallarán los protocolos identificados a partir de la información de requisitos.

Tabla 7-13 Plantilla de Protocolo

Nombre del Protocolo: *descripción breve de la interacción un*	
Iniciador: *rol o roles responsables de iniciar la interacción*	**Participante:** *rol o roles que interactúan con el iniciador*
Descripción: *descripción textual que explica el propósito el protocolo*	

Tabla 7-14 Protocolo *Bad Credit*

Nombre del Protocolo: Bad Credit	
Iniciador: Buyer Manager	**Participante:** Auctioneer
Descripción: Protocolo utilizado para informar de un intento de compra sin crédito suficiente en la cuenta.	

Tabla 7-15 Protocolo *Bid*

Nombre del Protocolo: Bid

Iniciador: Buyer	**Participante:** Auctioneer
Descripción: Rol para comunicar una puja por un producto.	

Tabla 7-16 Protocolo *Current Prince*

Nombre del Protocolo: Current Price	
Iniciador: Auctioneer	**Participante:** Buyer
Descripción: Informa a los compradores (Buyer) del precio actual al que se vende un producto, dentro de una ronda de subasta.	

Tabla 7-17 Protocolo *Communicate Sale*

Nombre del Protocolo: Communicate Sale	
Iniciador: Auctioneer	**Participante:** Buyer, Seller Manager
Descripción: Comunica una adjudicación de un producto y el precio por el cual se ha adjudicado.	

Tabla 7-18 Protocolo *Credit Update*

Nombre del Protocolo: Credit Update	
Iniciador: Buyer	**Participante:** Buyer Manager
Descripción: Envía una petición de actualización de crédito.	

Tabla 7-19 Protocolo *Exit*

Nombre del Protocolo: Exit	
Iniciador: Guest, Buyer, Seller	**Participante:** Buyer Manager, Seller Manager
Descripción: Comunica una adjudicación de un producto y el precio por el cual se ha adjudicado.	

Tabla 7-20 Protocolo *Open Account*

Nombre del Protocolo: Open Account	
Iniciador: Guest	**Participante:** Buyer Admitter, Seller

	Admitter
Descripción: Protocolo utilizado para solicitar la apertura de una cuenta de crédito en el sistema.	

Tabla 7-21 Protocolo *New Account*

Nombre del Protocolo: New Account	
Iniciador: Buyer Admitter, Seller Admitter	**Participante:** Buyer Manager, Seller Manager
Descripción: Notifica que se ha creado una nueva cuenta de crédito	

Tabla 7-22 Protocolo *Purchase*

Nombre del Protocolo: Purchase	
Iniciador: Buyer	**Participante:** Buyer Manager
Descripción: Notifica que se ha realizado una compra	

Tabla 7-23 Protocolo *New Round*

Nombre del Protocolo: New Round	
Iniciador: Auctioneer	**Participante:** Boss
Descripción: Solicita el inicio de una nueva ronda para subastar un producto.	

Tabla 7-24 Protocolo *Register*

Nombre del Protocolo: Register	
Iniciador: Guest	**Participante:** Buyer Admitter, Seller Admitter
Descripción: Envía una solicitud de registro o bien como Buyer o como Seller.	

Tabla 7-25 Protocolo Request *Good List*

Nombre del Protocolo: Request Good List

Iniciador: Auctioneer	**Participante:** Seller Admitter
Descripción: Solicita la lista de Productos que están preparados para subastarse.	

Tabla 7-26 Protocolo *Request Buyer List*

Nombre del Protocolo: Request Buyer List	
Iniciador: Auctioneer	**Participante:** Seller Admitter
Descripción: Solicita la lista de Buyer que se encuentran activos en el sistema y por consiguiente pueden participar en una ronda de subastas.	

Tabla 7-27 Protocolo *Sale Record*

Nombre del Protocolo: Sale Record	
Iniciador: Seller Manager	**Participante:** Seller
Descripción: Informa de Seller de que uno de los productos suyos ha sido vendido. Se envía la información de la compra (producto, precio, quién ha sido el comprador).	

Tabla 7-28 Protocolo *Submit Good*

Nombre del Protocolo: Submit Good	
Iniciador: Seller	**Participante:** Seller Admitter
Descripción: Protocolo utilizado para solicitar que un protocolo	

7.3.5 Reglas Organizacionales

A continuación se detallarán las reglas organizacionales detectadas para el caso de estudio. Se ha incluido una descripción textual de la regla y su tipo (regla de vivera o regla de seguridad). Para obtener el modelo de Reglas Organizacionales final deberían refinarse estas reglas añadiendo una formula en la notación propuesta por (Zambonelli, Jennings, & Wooldridge, 2001).

Tabla 7-29 Reglas Organizacionales generadas en Gaia

Regla	**Tipo**
El *Auctioneer* debería de esperar si no hay suficientes bienes a subastar	Viveza

El *Auctioneer* debe de esperar si no hay suficientes *Buyers* en la subasta	Viveza
Un *Buyer* no se puede entrar directamente en la sub-organización *Auction*	Seguridad
El *Buyer* debe de registrarse en la sub-organización de Registro antes de entrar en el sistema	Seguridad
Un *Buyer* no puede entrar en la sub-organización *Settelment* si no ha ganado una ronda	Seguridad

7.4 Conclusiones

En este capítulo se ha expuesto la resolución de un caso de estudio a aplicando RE4Gaia. En el punto 7.2 se muestra la aplicación de RE4Gaia, mientras que en el punto 7.3 se muestran los artefactos generados tras aplicar las transformaciones automáticas entre modelos.

Como caso estudio se seleccionó un mercado de pescado que funciona aplicando un protocolo de subasta. Este modelo fue propuesto en (Instituto de Investigación en Inteligencia Artificial (IIIA), 1998). Se escogió concretamente este caso de estudio porque muestra de una forma clara el funcionamiento un sistema compuesto por múltiples roles que buscan cumplir objetivos. Por otra parte, en la ingeniería del software, se considera que la *interacción* es probablemente la característica más importante de un software complejo (Wooldridge M. , An Introduction to Multi-Agent Systems, 2004). La aplicación este caso estudio ha dejado patente el funcionamiento de un sistema mediante múltiples componentes que interactúan entre sí y como nuestro enfoque permite identificar claramente estas interacciones a nivel de requisitos.

8 Conclusiones

En este último capítulo se presentarán las aportaciones del trabajo, las líneas futuras de investigación y las publicaciones obtenidas del trabajo de investigación realizado en esta tesis.

8.1 Aportaciones

En los últimos años se han propuesto múltiples metodologías para guiar el proceso desarrollo de software en los SMA. Una de las metodologías más importantes en este paradigma es Gaia, una metodología que permite el análisis y diseño de sistemas Multi-Agente tanto a nivel organizativo o social como a nivel individual de cada agente. La popularidad de Gaia se debe a su facilidad de aprendizaje y sencillez que permite realizar una especificación de un SMA de forma ágil. Sin embargo, a pesar de ser una de las metodologías más destacadas, Gaia no proporciona ninguna guía o soporte para la adquisición y modelado de requisitos.

El objetivo de esta tesis es aportar una solución a este problema, proponiendo un método de ingeniería de requisitos para Gaia satisfaciendo los sub-objetivos planteados al inicio del trabajo:

- Estudiar las soluciones existentes a la adquisición y modelado de requisitos en el campo de los sistemas Multi-Agente.
- Proponer un modelo que provea la expresividad necesaria al usuario para que pueda efectuar la adquisición y modelado de requisitos.
- Aplicar los principios de software dirigidos por modelo, dando soporte a los principios por la arquitectura *Model Driven Architecture* (Object Management Group, 2009). Este sub-objetivo incluirá la definición un meta-modelo de requisitos y un meta-modelo de la metodología Gaia.
- Definir las reglas de transformación de modelos para que puedan transformarse de forma sistemática la especificación de los requisitos del sistema en los modelos de análisis de la metodología Gaia.
- Especificar el Proceso de Desarrollo mediante un lenguaje de modelado de procesos de negocio, para especificar de una forma precisa como utilizar el método propuesto y los artefactos a ser generados.

Como punto de partida este trabajo, se realizó un análisis sobre el uso de las técnicas de Ingeniería de Requisitos en las metodologías de desarrollo de SMA. Para realizar el estudio se decidió aplicar una revisión sistemática de la literatura (Kitchenham, 2004), con el objetivo de determinar *qué* técnicas de Ingeniería de Requisitos se aplicaron en las metodologías Multi-Agente y *cómo* se aplicaron durante los últimos 10 años. Se escogió realizar este estudio esta forma, porque una revisión sistemática nos proporciona las ventajas de ser un método objetivo y repetible para la determinación de actividades de investigación relacionadas con un área de interés.

El estudio se llevó cabo mediante dos fases. En la primera se consideraban tres criterios para analizar los trabajos de investigación publicados en dos librerías digitales (ACM DL y IEEExplore). Posteriormente en (Blanes, Insfrán, & Abrahão, Requirements Engineering in the Development of Multi-Agent Systems: A Systematic Review, 2009) se realizó una revisión utilizando cuatro librerías digitales ACM Digital Library (ACM), IEEExplore Electronic Database (IEEExplore), Inspec (IE), y Science Direct (SD) analizando seis criterios y re-utilizando el trabajo hecho en la primera revisión.

Tras la aplicación del estudio se identificaron carencias en la investigación importantes. La mayor parte de las metodologías de SMA se centraron en el modelado y análisis de los requisitos y no en su adquisición. Para la adquisición la mayoría propuestas se basaron en i*. Éste hecho reveló la existencia de una carencia de métodos y técnicas alternativos para la adquisición de requisitos. Respeto a las notaciones se detectó un uso destacado de los modelos de roles. Otro aspecto importante, fue la falta de investigación sobre la traza habilidad entre los artefactos producidos durante el desarrollo de los SMA. La trazabilidad pre-requisitos solamente se aplicó en un trabajo. Por último, se detectó una falta de estudios empíricos para validar las propuestas. Los estudios empíricos son necesarios para aportar evidencias y determinar que técnicas son mejores en ciertas situaciones.

Las conclusiones extraídas de la revisión sistemática se tuvieron en cuenta para el desarrollo de la propuesta. Se decidió dividir la fase de tratamiento de requisitos en dos fases: una fase de modelado y un proceso de análisis. De este modo de adquirirán los requisitos con un modelo de mayor abstracción en la fase de modelado y posteriormente se refinarán en la fase de análisis tratándose a un nivel de detalle mayor.

Como punto de partida para iniciar la fase de modelado, se siguió un enfoque similar al propuesto en RETO (Insfran, 2003): una propuesta de modelado de requisitos para la obtención de esquemas conceptuales en el paradigma orientado a objetos. En RETO se parte de la *misión del sistema* para definir el propósito sistema, sus responsabilidades y su alcance. Esta misión del sistema se descompone aplicando la técnica de *árbol de refinamiento de funciones* con el objetivo de obtener las funciones elementales del sistema. Se decidió adoptar estados técnicas en el inicio del modelo requisitos de debido a que permitían establecer una jerarquía partiendo de una meta global a funciones más elementales. Esta idea es coherente con la metáfora organizacional propuesta en Gaia, donde el sistema tiene una misión global y a su vez está compuesto por múltiples sub-organizaciones con objetivos distintos, y estas a su vez compuestas por múltiples roles. En el último nivel del ARF se encuentran las funciones llevadas a cabo por estos roles.

Por otra parte dado el gran uso de los modelos de roles en este tipo sistemas, evidenciado en los resultados de la revisión sistemática, se decidió adoptar esta notación a un nivel alto abstracción. En consecuencia se incorporó un modelo de roles en la fase de modelado de requisitos. A este nivel le otorgamos al *rol* el significado de una entidad abstracta que provee múltiples funciones al sistema.

Por último se requiere modelar una entidad clave en los SMA: el *entorno*. Para identificar los recursos del entorno se decidió por la construcción de un modelo de dominio en la fase de modelado. Este modelo nos permite identificar las entidades claves en el dominio sus relaciones entre estas. Posteriormente, en la fase de análisis se refinarán estas entidades para obtener los recursos que finalmente formarán parte del entorno.

Tras la construcción de los modelos de la fase de *modelado requisitos*, se pasa a la fase de *análisis de requisitos*. En la fase anterior se identificaron las funciones que llevaban a cabo los roles. En los sistemas Multi-Agente, un agente puede realizar tareas por sí mismo o bien requerir de la participación de otro agente basándose en el comportamiento social de los agentes. Para realizar a cabo esta distinción se escogió la creación de diagramas de actividad.

Con estos diagramas se pretende modelar el flujo de actividad básicos los roles, permitiendo al analista detectar relaciones de dependencia entre estos roles.

Como último paso de la fase de análisis, se procede a la creación de las *reglas organizacionales*. Estas reglas en el objetivo de modelar a nivel global el comportamiento la organización y establecer restricciones sobre esta expresadas en lenguaje natural.

Uno de de los problemas detectados en las metodologías AOSE es el escaso soporte a la trazabilidad. Este hecho dificulta el mantenimiento de software y disminuye su calidad global. Para dar solución a estos problemas se optó por la aplicación de una arquitectura MDA al modelo de requisitos propuesto. Concretamente se construyen los meta-modelos tanto de RE4Gaia, como del análisis de Gaia, acorde a la arquitectura MOF. Como lenguaje de modelado de estos meta-modelos, nivel M3, se utilizó el Ecore una variante de *EMOF*. En el meta-modelo de RE4Gaia, nivel M2, se diseñaron los elementos necesarios para construir los modelos de RE4Gaia. De manera similar se procedió a definir el meta-modelo de Gaia en el nivel M2. Por último se definieron las transformaciones utilizando el lenguaje *QVT Relations*. La aplicación de la arquitectura MDA nos permite un convertir de forma automática una especificación en RE4Gaia, nivel M0, a un modelo de análisis en Gaia ejecutando las transformaciones en una herramienta que implemente el estándar QVT *Relations*.

El siguiente paso fue definir proceso desarrollo de software. La calidad del proceso de desarrollo utilizado, tiene una repercusión directa en la calidad del producto final. De forma que es necesaria una definición precisa de los procesos que se deben seguir para aplicar el método propuesto. Se utilizó como lenguaje de modelado de procesos SPEM, un meta-modelo propuesto como notación para la definición de procesos de desarrollo de software y sus componentes. Se seleccionó SPEM por la ventaja de ser un estándar propuesta por la OMG, siendo fácil su integración con el resto de sus estándares de esta, pudiéndose utilizar perfiles de UML para disfrutar de las herramientas que soportan el modelado en UML. En esta especificación se incluyeron las disciplinas en las que se propone el proceso; indicándose para cada una el conjunto de tareas, modelos y documentos generados, y las guías de construcción propuestas. Además se indicó en la descripción del proceso, las fases que componen el proceso y el orden en el que se deben de aplicar las tareas para aplicar nuestra propuesta. Además se incluyeron los modelos y documentos de entrada y de salida para tarea, permitiendo identificar las dependencias entre actividades y modelos y documentos.

Por último se validó la propuesta utilizando un caso de estudio: un mercado de pescado, que emplea un sistema de subasta. Se eligió este caso de estudio como apropiado porque muestra de una forma clara el funcionamiento un sistema compuesto por múltiples roles que buscan cumplir objetivos. De este como se comprobó que nuestro enfoque proporciona la expresividad suficiente para modelar los requisitos de un sistema compuesto por múltiples agentes que interactúan entre sí en una organización.

A continuación se resumen las aportaciones concretas de esta tesis:

- Se ha realizado un análisis sobre el uso de técnicas y notaciones para dar soporte a la Ingeniería de Requisitos en las metodologías de desarrollo de SMA.
- Se ha desarrollado una aproximación de ingeniería de requisitos, para dar soporte a la adquisición y modelado de requisitos en la metodología Gaia.
- Se han definido los meta-modelos de RE4Gaia y el modelo de análisis de Gaia.

- Se han definido transformaciones para que, mediante la ejecución de un motor de transformaciones, se puedan obtener los artefactos del análisis de Gaia a partir de los artefactos de RE4Gaia.
- Se ha definido utilizando SPEM un proceso de desarrollo para guiar la aplicación del método propuesto.
- Se ha validad la propuesta mediante la aplicación de un caso de estudio de un sistema de subasta en un mercado de pescado.

8.2 Trabajos Futuros

Destacamos cuatro líneas futuras investigación: desarrollo de un Modelo de requisitos no funcionales, integración dentro de un enfoque de Líneas de Producto, desarrollo de una herramienta CASE y realización de estudios empíricos para validar la propuesta.

En la primera línea de investigación, se trabajará en incorporar un modelo de requisitos no funcionales a la propuesta. Los requisitos no funcionales permiten especificar parámetros de calidad en el servicio. Se tiene previsto la aplicación de una Revisión Sistemática para determinar el uso de los requisitos no funcionales en las metodologías SMA. El fin de determinar qué atributos no funcionales son los más utilizados y extraer conclusiones del estudio para posteriormente construir un modelo de requisitos no funcionales. Por último se integrará el modelo tanto en el modelo de requisitos como en la metodología Gaia. Se tiene previsto realizar la integración en un marco de *Desarrollo de Software Dirigido por Modelos*.

En la segunda línea de investigación, se propone integrar y ampliar el enfoque de requisitos propuestos dentro de un marco de desarrollo de Líneas de Producto de Software. Las Líneas de Producto Software (LPS) son un enfoque de desarrollo de software que promueve la reutilización de activos en una familia de productos identificando y especificando aquellas características que son comunes y variables en la familia de productos.

En la tercera línea de investigación abierta, se desea crear una herramienta gráfica que de soporte a los modelos propuestos en RE4Gaia. Una de las opciones que se plantean es utilizar el *Eclipse Graphical Modeling Framework (GMF)* (The Eclipse Foundation, 2009). GMF permite a los desarrolladores crear editores gráficos basados en EMF, de forma que podrían construirse editores gráficos para los meta-modelos previamente generados. La creación de una herramienta CASE facilitaría al usuario el uso de la herramienta, al hacer la especificación más intuitiva y fácil de entender.

En la última línea de investigación se pretende desarrollar experimentos para validar nuestra propuesta. Los estudios experimentales son un método efectivo para proveer evidencia empírica sobre la mejora y entendimiento de los métodos y técnicas de la ingeniería del software. En nuestro caso, se diseñaran y ejecutarán experimentos para analizar la eficacia y entendibilidad del método propuesto para guiar el proceso de modelado de requisitos, obteniéndose retroalimentación para la mejora del mismo.

Para concluir, estas líneas de investigación futuras tienen previsto realizarse en el marco del proyecto CICYT (*MULTIPLE: Multimodeling Approach for Quality-Aware Software Product Lines*) del grupo de investigación *ISSI*.

8.3 Resultados Obtenidos

En esta sección se presentan las publicaciones relacionadas con esta tesis. Las tres publicaciones relacionadas fueron publicadas en Congresos Internacionales relacionados con los Sistemas Multi-Agente y con un estricto proceso de revisión.

- David Blanes, Emilio Insfrán, Silvia Abrahão: *Reviewing the Use of Requirements Engineering Techniques in the Development of Multi-Agent Systems*. 2nd International Symposium on Distributed Computing and Artificial Intelligence 2009, 10th-12th June, 2009, Salamanca, Spain, S. Omatu et al. (Eds.): IWANN 2009, Part II, Lecture Notes in Computer Science (LNCS) 5518, pp. 134–137, Springer-Verlag Berlin Heidelberg 2009

- David Blanes, Emilio Insfrán, Silvia Abrahão: *Requirements Engineering in the Development of Multi-Agent Systems: A Systematic Review*. 10th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2009), 23rd-26th September, 2009, Burgos, Spain, Lecture Notes in Computer Science (LNCS) 5788, pp. 510-517, Berlin Heidelberg, Springer, ISBN 3-642-04393-3.

- David Blanes, Emilio Insfrán, Silvia Abrahão: *RE4Gaia: A Requirements Modeling Approach for the Development of Multi-Agent Systems*. International Conference on Advanced Software Engineering and Its Applications ASEA 2009, 10th – 12nd December, 2009, Jeju Island, Korea, Proceedings Series: Communications in Computer and Information Science, Vol. 59, pp. 245-252, Berlin Heidelberg, Springer, ISBN: 978-3-642-10618-7.

Anexo A. Trabajos seleccionados en la Revisión Sistemática

Fuente	Referencia
ACM DL	1. "Design and Evaluation of a Multiagent Autonomic Information System". W. H. Oyenan, S. A. DeLoach.. IAT 2007. 2. "Reasoning about Willingness in Networks of Agents". S. Dehousse, S Faulkner, H. Mouratidis, P. Giorgini, M. Kolp. SELMAS 2006 3 "Towards Goal-Oriented Development of Self-Adaptive Systems". M. Morandini, L. Penserini, A. Perini. SEAMS 2008. 4. "An Adaptive Multi-Agent Organization Model Based on Dynamic Role Allocation". M. Hoogendoorn, J. Treur. IAT 2006. 5. "Modelling Secure Multiagent Systems". H. Mouratidis, P. Giorgini, G. Hanson. AAMAS 2003. 6. "From Requirements to Multi-Agent Architecture Using Organisational Concepts". L. Bastos, J. Castro. SELMAS 2005. 7. "Information Systems Development through Social Structures". M. Kolp, P. Giorgini, J. Mylopoulos. SEKE 2002. 8. "Enforcing a Security Pattern in Stakeholder Goal Models". Y.Yu, H.Kaiya, H. Washizaki. QoP. 2008. 9. "Requirements-Driven Design of Autonomic Application Software". A. Lapouchnian, Y. Yu, S. Liaskos, J. Mylopoulos. CASCON 2006. 10. "BDI-Agents for Agile Goal-Oriented Business Processes". B. Burmeister, M. Arnold, F. Copaciu, G. Rimassa. AAMAS 2008. 11. "Integrating Scenarios, i*, and AspectT in the Context of Multi-Agent Systems". A. P. A. Oliveira, L. M. Cysneiros, J. C. S. P. Leite, E. M. L. Figueiredo, C. J. P. Lucena. CASCON 2006 12. "A New Approach to the BDI Agent-Based Modeling". C.-H. Jo, G. Chen, J. Choi. SAC 2004. 13. "Goal-Oriented Requirement Analysis for Data Warehouse Design". P. Giorgini, S. Rizzi, M. Garzetti. DOLAP 2005. 14. "Peer-to-peer Autonomic Location Based Services". P. Leong, S. Sukumar, V. Lee. Mobility 2007 15. "Intelligent Agents for QoS Management ". K. Trzec, D. Huljenic. AAMAS

2002.

16. "A Hybrid Model for Agent Based System Requirements Analysis". P. Ranjan, A. K. Misra. ACM SIGSOFT Software Engineering Notes, 31(3). 2006.

17. "High Variability Design for Software Agents: Extending Tropos". L. Penserini, A. Perini, A. Susi. Transactions on Autonomous and Adaptive Systems (TAAS) , Volume 2 Issue 4. 2007.

18. "Improving the Agent-Oriented Modeling Process by Roles". R. Depke, R. Heckel, J. M. Küster. AGENTS 2001.

19. "SPEM on test: the SODA case study". E. Nardini, A. Molesini, A. Omicini, E. Denti. SAC 2008

20. "Goal-Oriented Development of BDI Agents: the PRACTIONIST Approach". V. Morreale, S. Bonura, G. Francaviglia, F. Centineo. IAT 2006.

21. "A Knowledge Level Software Engineering Methodology for Agent Oriented Programming". P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos. AGENTS 2001.

22. "Agent Based System Development – A Domain-Specific Goal Approach". P. Ranjan, A. K. Misra. ACM SIGSOFT Software Engineering Notes, 31(3). 2006.

23. "Modeling Mental States in the Analysis of MAS Requirements" A. Lapouchnian, Y. Lespérance. AOSE 2006.

24. "Multiagent Systems Engineering of Organization-based Multiagent Systems". Scott A. DeLoach. SELMAS 2005.

25. "A Product-Line Requirements Approach to Safe Reuse in MAS " J. Dehlinger, R. R. Lutz. SELMAS 2005.

26. "Information Systems as Social Structures". A. Fuxman, P. Giorgini, M. Kelp, J. Mylopoulos, FOIS 2001.

27. "A Formal Approach to Agent Design: An Overview of Constraint-Based Agents" A. K. Mackworth, Y. Zhang. Constraints, 8(3), 229–242. 2003.

28. "A HELIX-SPINDLE MODEL for Ontological Engineering". R. Kishore, H. Zhang, R. Ramesh. Communications of the ACM , 47(2). 2004

29. "A Seamless Approach to the Agent Development". C.-H. Jo. SAC 2001.

30. "Tropos: An Agent-Oriented Software Development Methodology". P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos. Autonomous

	Agents and Multi-Agent Systems, 8(3). 2004.
IEEE Xplorer	31. "A Method for Agent-Based System Requirements Analysis ". C.-H. Leon Lee, A. Liu. MSE 2002. 32. "Integrative Early Requirements Analysis for Agent-Based Systems". L. Cao, C. Zhang, D. Luo, W. Chen, N. Zamani. HIS 2004. 33. "Towards a formal methodology for developing multi-agent applications using temporal Z ". A. Regayeg, A. H. Kacem, M. Jmaiel. Computer Systems and Applications 2005. 34. "Brainmerge: a Semiotic-Oriented Software Development Process for Intelligence Augmentation Systems ". A.L.O. Paraense, R.R. Gudwin, R.A. Gonçalves. KIMAS 2007 35. "A Novel Approach of Requirements Analysis for Agent Based System ". P. Ranjan, A. K. Misra. SNPD 2006. 36. "Guiding agent-oriented requirements elicitation: HOMER". D. Wilmann, L. Sterling. QSIC 2005. 37. "Security and Privacy Requirements Analysis within a Social Setting". L. Liu, E. Yu, J. Mylopoulos. RE 2003. 38. "Tropos: A Requirements-Driven Methodology for AOS" P. Giorgini, M. Kolp, J. Mylopoulos, J. Castro. 2003 39. “Requirements Elicitation and Analysis of Multiagent Systems Using Activity Theory”. R. Fuentes-Fernández, J.J. Gómez-Sanz, J. Pavón. IEEE Transactions on Systems, Man and Cybernetics, 39(2). 2009
Libro	40. "The Agent-Oriented Methodology MAS-CommonKADS". C.A. Iglesias, M. Garijo. 41. "From Requirements to Code with the PASSI Methodology". M. Cossentino 42. "Prometheus: A Practical Agent-Oriented Methodology". L. Padgham, M. Winikoff 43. "MAS as Computational Organizations: The Gaia Methodology" F. Zambonelli, N. R. Jennings, M. Wooldridge 44. "Engineering Adaptative MAS: The ADELFE Methodology". C. Bernon, V. Camps, M.-P. Gleizes, G. Picard 45. "The MESSAGE Methodology for Agent-Oriented Analysis and Design". F. J. Garijo, J. J. Gómez-Sanz, P. Massonet.

	46. "The INGENIAS Methodology and Tools". J. Pavón, J. J. Gómez-Sanz, R. Fuentes. 47. "Towards Radical Agent-Oriented Software Engineering Processes Based on AOR Modelling". K. Taveter, G. Wagner 48. "Multi-Agent Systems Engineering: An Overview and Case Study". S. A. Deloach, M. Kumar
Inspect	49. "A combined specification language and development framework for agent-based application engineering". J.A.R.P. Sardinha, R. Choren, V.T. Silva., R. Milidiú., C. J.P. Lucena. The Journal of Systems and Software 79, 1565–1577. 2006 50. "Formal agent-oriented modeling with UML and graph transformation". R. Depke, R. Heckel , J. M. Küster. Science of Computer Programming 44 229 – 252. 2002
Science Direct	51. "Development of intelligent multisensor surveillance systems with agents". J. Pavón , J. Gómez-Sanz, A. Fernández-Caballero, J. Valencia-Jiménez. Robotics and Autonomous Systems 55 (2007) 892–903. 2007 52. "MOBMAS: A methodology for ontology-based multi-agent systems development". Q.-N. N. Tran, G. Low. Information and Software Technology 50 (2008) 697–722. 53. "Adaptive Agent Model: Software Adaptivity using an Agent-oriented Model-Driven Architecture". L. Xiao, D. Greer. Information and Software Technology 51 (2009) 109–137 54." Consistency preserving co-evolution of formal specifications and agent-oriented conceptual models". A. Krishna, S. A. Vilkomir, A. K. Ghose. Information and Software Technology 51 (2009) 478–496 55. "The Agent–Object-Relationship metamodel: towards a unified view of state and behavior". G Wagner. Information Systems 28 (2003) 475–504 56. "Towards requirements-driven information systems engineering: the Tropos project". J. Castro, M. Kolp, J. Mylopoulos. Information Systems 27 (2002) 365–389 57. "Engineering the social: The role of shared artifacts". J. Paay , L. Sterling, F. Vetere , S. Howard , A. Boettcher. Int. J. Human-Computer Studies 67 (2009) 437–454 58. "Goal-oriented requirements analysis and reasoning in the Tropos methodology". P. Giorgini, J. Mylopoulos, R. Sebastiani. Engineering Applications of Artificial Intelligence 18 (2005) 159–171

Anexo B. Especificación de los meta-modelos

En esta sección presentaremos los meta-modelos propuestos para nuestro modelo de requisitos RE4Gaia y para el meta-modelo de Análisis destino, Gaia.

Introducción

En esta sección se presentaran los conceptos empleados en la descripción de las clases del meta-modelo RE4Gaia.

Descripción

Se incluye una definición informal de la meta-clase. En esta sección también se puede indicar si una meta-clase es abstracta.

Asociaciones

Es listan los fines de asociación opuestos que conectan con la clase. Se puede indicar la multiplicidad de un fin de asociación.

Adicionalmente, para las Enumeraciones se mostrará el siguiente apartado:

Enumeraciones

Contiene una lista con todos los valores de la enumeración con el siguiente formato:

- *Literal*.

  Descripción.

Donde *literal* es el valor del literal, y *descripción* es una breve explicación del literal.

Meta-modelo RE4Gaia

En esta sección se listaran todos los componentes que conforman el meta-modelo RE4Gaia. En este meta-modelo están contenidos los elementos de un proyecto en RE4Gaia. Se incluyen todos los elementos para modelar los artefactos tanto de la fase de Modelado de Requisitos como de la fase de Análisis de Requisitos. El paquete metaRE4Gaia actuará como paquete contenedor de todos estos elementos. Adicionalmente, para cada diagrama se han agrupado sus meta-clases en paquetes, para facilitar la comprensión del meta-modelo y favorecer la mantenibilidad.

Clase Project

Descripción

La clase Project incluye todos los elementos necesarios para instanciar un proyecto.

Asociaciones

- Diagrams: Diagram [0..*]
  Referencia a los diagramas incluidos en el proyecto.

Clase Model

Descripción

Esta meta-clase contiene información común entre los distintos modelos que conforman un Proyecto. Es de tipo Abstracto dado que su objetivo no es ser instanciada directamente, sino reflejar el conjunto de información entre modelos.

Asociaciones

- project : Project [1]
  Referencia al Proyecto donde está contenido el Modelo.

Clase Element

Descripción

Meta-clase abstracta que contiene información estructural común de elementos del tipo “Element”.

Esta meta-clase se utiliza para modelar información común entre algunos artefactos del meta-modelo RE4gaia: identificador, nombre el elemento y descripciones.

Asociaciones

No existen asociaciones adicionales.

Paquete mission Statement

Este paquete contiene los elementos necesarios para especificar la declaración de la misión del sistema.

Clase mission Statement

Descripción
Esta clase contiene la información de la Misión del Sistema de un Proyecto.

Asociaciones
No existen asociaciones adicionales.

Paquete functionalRefinementTree

En este paquete se incluyen los elementos para representar un Árbol de Refinamiento de Funciones.

Clase Functional Requirement Tree

Descripción
Esta meta-clase represente a una meta-clase contenedora de los elementos de un Árbol de Refinamiento de Funciones.

Asociaciones
- suborganizations : Suborganization [0..*]
  Referencia a las Sub-organizaciones que componen el diagrama. El atributo Contaiment tendrá el valor true, indicando que la clase actúa como contenedor de las sub-organizaciones.

Clase Suborganization

Descripción
Esta meta-clase contiene la información para representar a una sub-organización. Hereda de *Element*, atributos necesarios como el identificador, el nombre y la descripción.

Asociaciones
- roles: Role [0..*]

  Conjunto de roles que componen la sub-organización.

- activity Diagrams: Activity Diagram [0..*]

  Conjunto de Diagramas de Actividad contenidos en la sub-organización.

- Frt: Functional Refinement Tree [1]

  Árbol de Refinamiento de Funciones que contiene la sub-organización.

Clase Role

Descripción
Esta meta-clase especifica un Rol perteneciente a una sub-organización.

Asociaciones

- functions : Function [0..*]
  Referencias al conjunto de funciones que tiene que desempeñar un rol en una sub-organización.
- roleModel : RoleModel [0..1]
  Si este rol ha sido incluido en un modelo de roles, se almacena la referencia en este campo.

Clase Function

Descripción

Modela una función desempeñada por un Rol.

Asociaciones

- activities : Activity [0..*]
  Referencias al conjunto de funciones que tiene que desempeñar un rol en una sub-organización.

Paquete requirements Role Model

Este paquete contiene las meta-clases para modelar el Modelos de Roles de la fase de Requisitos.

Clase Requirements Role Model

Descripción

Este elemento represente un Modelo de Roles, compuesto por un conjunto de Roles y un conjunto de Generalizaciones entre estos Roles.

Asociaciones

- roles : Role [0..*]
  Referencias a los Roles que serán incluidos en este modelo.
- generalizations: RoleGeneralization [0..*]
  Conjunto de Generalizaciones que se incluyen en el diagrama.

Clase Role Generalitation

Descripción

Define una relación de generalización de un rol a otro.

Asociaciones

- from: Role [1]

  Elemento que es una especialización del rol referenciado en To.

- to: Role [1]

  Elemento que es una generalización del rol referenciado en From.

- requirementsRoleModel [1]

Referencia al modelo de roles que incluye esta generalización.

Paquete domain Model

Este paquete contiene las meta-clases y relaciones entre ellas para modelar un Modelo de Dominio.

Clase Domain Model

Descripción

Esta clase representa a un modelo de Dominio identificado en la fase de modelado de requisitos de RE4Gaia.

Asociaciones

- entities: Entity [0..*]

Entidades contenidas en el Modelo de Dominio.

- relations: Relation [0..*]

Conjunto de relaciones contenidas en el Modelo de Dominio.

Clase Entity

Descripción

Esta meta-clase modela una entidad del Modelo de Dominio.

Asociaciones

- domainModel: Domain Model [1]

Modelo de dominio al que pertenece la entidad.

Clase Relation

Descripción

Una relación hace referencia a dos entidades relacionadas. Una relación es una meta-clase abstracta.

Asociaciones

- from: Entity [0..1]

Entidad desde donde parte una relación.

- to: Entity [0..1]

Entidad donde llega una relación.

- domainModel: DomainModel [0..1]

Modelo de Dominio al que pertenece la relación.

Clase Generalization

Descripción

Una Generalización es una relación taxonómica entre una entidad más general y una más específica. Cada instancia de la entidad más específica es también una instancia indirecta de la clase más general. De este modo, la entidad más específica hereda las características de la entidad más general.

Asociaciones

No existen asociaciones adicionales.

Clase Association

Descripción

Una Asociación especifica una relación semántica que puede ocurrir entre dos instancias tipadas.

Asociaciones

No existen asociaciones adicionales.

Paquete Activity Diagram

En este paquete se incluyen las meta-clases y relaciones entre ellas necesarias para especificar todos los elementos de un Diagrama de Actividad en la fase de Análisis de Requisitos en RE4Gaia.

Clase Activity Diagram

Descripción

Esta meta-clase actúa como contenedor para los elementos de un diagrama de actividad.

Asociaciones

- partitions: Activity Partition [0..*]

Particiones contenidas en el diagrama de actividad.

- suborganization: Sub-Organization [1]

Sub-organización que contiene al diagrama de actividad.

Clase Activity Partition

Descripción

Especifica una partición del diagrama de actividad. Esta partición modela el comportamiento de un rol.

Asociaciones

- role: Role [1]

  Hace referencia al Rol que se represente en esta partición.

- nodes: Activity Node [0..*]

  Nodos de actividad contenidos en la partición

- Edge: Activity Edge [0..*]

  Artistas contenidas en la partición.

- Diagram: Activity Diagram [1]

  Diagrama al que pertenece la partición.

Clase Comment

Comentario para especificar una restricción en lenguaje natural.

Descripción

Los comentarios se utilizan para especificar restricciones en el comportamiento de un rol dentro de un diagrama de actividad.

Asociaciones

- Partition: Activity Partition [0..1]

  Partición en la que está incluido el comentario.

Clase Activity Node

Meta-clase abstracta, cuyas instancias representan puntos del flujo de actividad conectados por aristas.

Descripción

Un nodo de actividad es una meta-clase abstracta que representa los pasos de una actividad. Cubre nodos de actividad y nodos decontrol.

Asociaciones

- Incoming: Activity Edge [0..*]

  Aristas que tienen el Nodo como destino.

- Outgoing : Activity Edge [0..*]

Aritas que tienen el Nodo como origen.

- partition : Activity Partition [0..1]

Partición en la que está contenido el nodo.

Clase Activity Edge

Un Activity Edge es una meta-clase abstracta para conexiones directas entre dos nodos de actividad.

Descripción

Esta meta-clase abstracta representa una arista entre dos nodos de actividad. Cubre nodos de control y nodos de decisión.

Asociaciones

- source: Activity Node [0..*]

Nodos que llegan la arista.

- target : Activity Node [0..*]

Nodos que alcanza la arista.

- partition : Activity Partition [0..1]

Partición en la que está contenida la arista.

Clase Activity

Descripción

Una actividad representa una actividad llevaba a cabo por un rol. La actividad la realiza un rol sin la cooperación con otros roles.

Asociaciones

- functions : Function [0..*]

Función del Árbol de Refinamiento de Funciones relacionado con dicha Actividad.

Clase Control Node

Descripción

Un nodo de control es un nodo de actividad utilizado para controlar el flujo entre otros nodos. Cubre: nodos iniciales, nodos finales, bifurcaciones, uniones y nodos de decisión.

Asociaciones

No existen asociaciones adicionales.

Clase Initial Node

Descripción

Un *Initial Node* es un nodo de control cuyo flujo empieza cuando se invoca a la actividad. Una actividad puede tener más de uno nodo inicial.

Asociaciones

No existen asociaciones adicionales.

Clase Final Node

Descripción

Un Final Node destruye todos los flujos de actividad que llegan a él.

Asociaciones

No existen asociaciones adicionales.

Clase Fork Node

Descripción

Un *Fork Node* es un nodo de control que divide un flujo en varios flujos concurrentes. Un *Fork Node*, tiene una arista entrante y múltiples aristas de salida.

Asociaciones

No existen asociaciones adicionales

Clase Join Node

Un *Join Node* es un nodo de control que sincroniza flujos múltiples.

Descripción

Un *Join Node*, tiene una arista entrante y múltiples aristas de salida.

Asociaciones

No existen asociaciones adicionales.

Clase Decision Node

Descripción

Un *Decision Node*, tiene una arista entrante y múltiples aristas de salida. Hacia donde se dirigirá el flujo de entre las múltiples aristas de salida, depende de una condición de evaluación.

Asociaciones
No existen asociaciones adicionales

Clase Control Flow

Descripción
Meta-clase abstracta que representa un nodo de control. Un *Control Flow* es una arista que empieza un nodo de actividad cuando el nodo predecesor ha finalizado.

Asociaciones
No existen asociaciones adicionales.

Clase Decision Flow

Descripción
Un *Decision Flow* contiene información sobre la guarda. Esta forma un *Decision Node* elegirá por cuál de los posibles *Decision Flow* debe continuar dependiendo de la evaluación de las guardas.

Asociaciones
No existen asociaciones adicionales.

Clase Send Signal Action

Descripción
SendSignalAction es una acción que crea una instancia de una señal y la trasmite al objeto destino. Representa una acción que envía una señal a un objeto destino. Se utiliza para modelar el concepto de iniciación de protocolo de RE4Gaia.

Asociaciones
No existen asociaciones adicionales.

Clase Accept Event Action

Descripción
Esta acción espera hasta que recibe una señal desde un objeto SendSignalAction. Modela el comportamiento de respuesta a un protocolo en RE4Gaia.

Asociaciones
No existen asociaciones adicionales.

Paquete Organizational Rules Model

Este paquete contiene los elementos necesarios para representar un Modelo de Reglas Organizacionales.

Clase Organizational Rules Model

Descripción

Un modelo de reglas organizacionales es un conjunto de reglas que modelan el comportamiento de una organización a nivel global.

Asociaciones

- Rules: Rule [0..*]

  Conjunto de reglas contenidas en el modelo.

Clase Rules

Descripción

Especifica una regla organizacional. Estas reglas pueden ser *liveness* o *security*. Las reglas *Liveness* modelan un modelo deseado en el sistema. Las reglas del tipo Security especifican restricciones que deben cumplirse en todo momento en el sistema.

Asociaciones

- rulesModel: Organizational Rules Model [1]

  Modelo de reglas que contiene la regla.

Paquete Environmental Model

Este paquete contiene los elementos necesarios para especificar un Modelo de Entorno. Aquí se muestran los recursos del sistema y los permisos mediante los cuales pueden ser accedidos por los roles.

Clase Environmental Model

Descripción

Esta meta-clase contiene los elementos para representar un modelo de entorno: un conjunto de recursos y un conjunto de recursos, utilizados para que los roles puedan acceder a los recursos.

Asociaciones

- Resources: Resource [0..*]

  Recursos a los que pueden acceder los roles.

- Permissions: Permission [0..*]

Permisos utilizados por los recursos para acceder a los roles.

Clase Resource

Descripción

Representa un recurso del entorno al que puede acceder un rol mediante un permiso.

Asociaciones

- Entity: Entity [0..*]

Entidad proveniente del modelo de dominio y relacionada con el recurso.

- Environment: Environmental Model [1]

Modelo de Entorno que contiene al recurso.

Clase Permission

Permiso utilizado por un rol para acceder a un recurso.

Descripción

Modela como acceder un rol a un determinado recurso.

Asociaciones

- Role: Role [1]

Rol que accede al recurso.

- Resource: Resource [1]

Recurso que es accedido.

- Environment: Environmental Model [1]

Modelo de Entorno que contiene el permiso.

Paquete enumerates

Este paquete contiene las Enumeraciones del meta-modelo RE4Gaia. A continuación se describen las enumeraciones que lo forman.

Enumeración Permission Type

Esta enumeración muestra los tipos de permisos con los que puede acceder un rol a un recurso.

Literales

- Read.

Permiso de lectura. No permite modificarlo.

- Modify

Permiso para modificar el recurso.

- Consume

Permiso para consumir el recurso.

Enumeración Rule Type

Enumera el tipo de reglas que se contemplarán en el sistema.

Literales

- Liveness.

Modela un comportamiento deseado en el sistema.

- Safety.

Representa una restricción del sistema. Es un invariante que debe cumplirse durante todo el tiempo.

Meta-modelo Gaia

Paquete metaGaia

Este paquete incluye todos los elementos necesarios para especificar los modelos de análisis en la metodología Gaia

Clase System

Descripción

La clase *System* incluye todos los elementos necesarios para instanciar un sistema.

Asociaciones

- diagrams: Diagram [0..*]
  Referencia a los diagramas incluidos en el sistema.

Clase Model

Descripción

Es de tipo Abstracto dado que su objetivo no es ser instanciada directamente, sino reflejar el conjunto de información entre diagramas.

Asociaciones

- system : System [1]
  Referencia al Sistema donde está contenido el Modelo.

Clase Element

Descripción

Meta-clase abstracta que contiene información estructural común de elementos del tipo "*Element*".

Asociaciones

No existen asociaciones adicionales.

Paquete subOrganizations

En este paquete están contenidos los elementos para representar el conjunto se sub-organizaciones que forman un sistema en Gaia.

Clase Organization

Descripción

Meta-clase que representa a una estructura organizacional en Gaia.

Asociaciones
No existen asociaciones adicionales.

Clase Suborganization

Descripción
De acuerdo con (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003) podemos considerar una organización cuando encontramos porciones del sistema global que:

i. Exhiben un comportamiento específicamente orientado para conseguir un sub-objetivo dado.

ii. Interactúan débilmente con otras porciones del sistema.

iii. Requieren competencias que no se requieren en otras partes del sistema.

Asociaciones
- organization: Organizations [1]

Referencia al Modelo de Sub-Organziaciones que contiene a la sub-organización.

Paquete environment

Paquete que contiene los elementos necesarios para representar un Modelo de Entorno.

Clase Environment

Descripción
Esta clase modela la representación abstracta del entorno donde se sitúa el sistema Multi-Agente.

Asociaciones
- resources: Resource [0..*]

Conjunto de recursos que forman el Modelo de Entorno.

Clase Resource

Descripción
Son recursos computacionales abstractos que un agente puede leer, modificar o consumir.

Asociaciones
- environment: Environment [0..1]

Entorno al que pertenece el recurso actual.

Paquete roleModel

En este paquete están contenidas las meta-clases y sus respectivas relaciones entre ellas para especificar un Modelo de Roles.

Clase RoleModel

Descripción

Representa el modelo de roles de una sub-organización, el cual estará formado por un conjunto de roles.

Asociaciones

- roles: Role [0..*]

  Conjunto de roles que componen el modelo.

Clase Role

Descripción

De acuerdo con con Zambonelli (Zambonelli, Jennings, & Wooldridge, Developing Multiagent Systems:The Gaia Methodology, 2003), un rol es un conjunto bien definido de responsabilidades o sub-objetivos en el contexto del sistema global.

Asociaciones

- Activities: Activity [0..*]

  Conjunto de actividades que desempeña el rol.

- Initiates: Protocol [0..*]

  Protocolos que inicia el rol.

- isPartner: Protocol [0..*]

  Protocolos en los que el role es el respondedor.

- permission: Permission [0..*]

  Conjunto de permisos que posee el rol sobre los recursos del sistema.

- responsabilities: Responsibility [0..*]

  Conjuntos de responsabilidades que posee el rol. Pueden ser de viveza o seguridad.

- roleModel: RoleModel [1]

  Modelo de roles donde está contenido el rol.

Clase Activity

Actividad que un agente o rol puede realizar.

Descripción

Unidad de acción que un agente lleva a cabo y que no precisa de la interacción con otro agente.

Asociaciones

- role: Role [0..*]

  Rol al que pertenece la actividad.

Clase Responsibility

Especifica una responsabilidad perteneciente a un rol o agente.

Descripción

Determina el comportamiento esperado de un rol. Este tipo de meta-clase es abstracta, de forma que no puede instanciarse directamente.

Asociaciones

- role: Role [0..1]

Rol al que pertenece la responsabilidad

Clase LivenessResponsibility

Descripción

Definen las trayectorias de ejecución potenciales a través de actividades e interacciones asociadas con el rol.

Los elementos atómicos de una expresión "*liveness*" son actividades o protocolos. Estos a su vez se ensamblan mediante los conectores mostrados en la Tabla B-1, formándose expresiones "*liveness*".

Tabla B-1 Operadores *liveness*

Operador	Interpretación
x.y	x seguido por y
X | y	Ocurrirán x o y
X*	x ocurre cero o más veces
X+	X ocurre una o más veces
x^{ω}	X ocurre infinitamente
[x]	X es opcional
X || y	X e y intercaladas

Asociaciones
No existen asociaciones adicionales.

Clase SecurityResponsibility

Descripción
Aseguran que "nada malo pasa", es decir que nos mantenemos en estados aceptables. Son invariantes que se tienen que cumplir en todos los estados de la ejecución del sistema.

Se utilizan simplemente predicados para especificar propiedades de seguridad. Se recomienda que se expresen sobre variables de permisos del rol. Como por ejemplo en la expresión:

number of papers = number of review forms

Asociaciones
No existen asociaciones adicionales.

Clase Permission

Descripción
Establecen los límites sobre hasta cuando un rol puede operar.

Asociaciones
- resource: Resource [0..1]
  Recurso al que se accede con el permiso
- role: Role [0..1]
  Rol autorizado a utilizar el permiso.

Paquete interactionModel

Contiene los elementos de un Modelo de Interacciones en Gaia.

Clase InteractionModel

Descripción
Representa a un modelo de interacción de Gaia, el cual está compuesto de protocolos.

Asociaciones
- Protocols: Protocol [0..*]

  Protocolos que conforman el Modelo de Interacciones.

Clase Protocol

Descripción

Los protocolos son actividades que requieren la interacción con otro agente para conseguir llevar a cabo la acción.

Asociaciones

- Initiator : Rol [0..*]

  Rol o roles que inician un protocolo.

- Responder : Rol [0..*]

  Rol o roles con los que interactúa el rol / los roles iniciadores del protocolo.

Paquete organizationalRules

Contiene los elementos para construir un Modelo de Reglas Organizacionales.

Clase OrganizationalRulesModel

Descripción

Las instancias de esta meta-clase especifican un Modelo de Reglas Organizacional.

Asociaciones

- Rules: Rule [0..*]

  Reglas que forman el Modelo de Reglas Organizacionales.

Clase OrganizationalRule

Descripción

Se consideran responsabilidades de la organización, considerando a esta como un todo. Esta meta-clase es abstracta, de modo que no puede ser instanciada directamente.

Asociaciones

- rulesModel: OrganizationalRulesModel [1]

  Modelo de Reglas que contiene la regla.

Clase LivenessRule

Descripción

Una regla de viveza define como ha de evolucionar la dinámica de la organización a lo largo del tiempo

Asociaciones
No existen asociaciones adicionales.

Clase SafetyRule

Descripción
Definen invariantes en el tiempo que la organización debe respetar.

Asociaciones
No existen asociaciones adicionales.

Paquete enumerates

Este paquete contiene las Enumeraciones del meta-modelo de Gaia.

Enumeración PermissionKind

Esta enumeración muestra los tipos de permisos con los que puede acceder un rol a un recurso.

Literales
- Read.

  Permiso de lectura. No permite modificarlo.

- Modify

  Permiso para modificar el recurso.

- Consume

  Permiso para consumir el recurso.

Anexo C. Transformaciones de Modelos

A continuación de mostrarán las Transformaciones de Modelos especificadas en el lenguaje *QVT Relations* para transformar automáticamente una especificación de requisitos en RE4Gaia en un modelo de análisis en Gaia.

```
transformation RE4GaiaToGaia(source: metaGaia, target: metaGaia){
      /* Key definition */
      key metaRE4Gaia::functionalRefinementTree::Role {name};
      key metaRE4Gaia::interactionModel::Protocol {name};
      key metaRE4Gaia::environment::Resource {name} ;
      key metaRE4Gaia::organizationalRulesModel::Rule{description};
      key metaRE4Gaia::functionalRefinementTree::Function {name};

      /* Keys for Gaia metamodel */
      key metaGaia::organizationalRulesModel::Rule{description};
      key metaGaia::roleModel::Role {name};
      key metaGaia::roleModel::Activity {name, role};
      key metaGaia::environment::Resource {name};
      key metaGaia::environment::Resource {name};
      key metaGaia::interactionModel::Protocol {name};

      /* relation metaGaia */
      top relation ProjectMapping {

        varName : String;
        varDescription : String;

            checkonly domain source m1:metaRE4Gaia::Project  {
             name = varName, description = varDescription
            };

            enforce domain target m2:metaGaia::System  {
             name = varName, description = varDescription
            };
      }

      /* Mission Statement Mapping */
      top relation missionStatementMapping {
            varMsg : String;
            checkonly                  domain                  source
m1:metaRE4Gaia::missionStatement::MissionStatement{
                  description = varMsg,
                  project =  p: metaGaia::Project {}

            };
            enforce domain target m2:metaGaia::System{
                  goal = varMsg
            };
            when {
             ProjectMapping(p,m2);
            }
      }
```

```
      /* Organizations Mapping */
      top relation OrganizationalMapping {

            checkonly                    domain                    source
domain1:metaRE4Gaia::functionalRefinementTree::FunctionalRefinementTre
e  {
              project = p:metaGaia::Project{}
            };

            enforce                      domain                    target
domain2:metaGaia::organizationModel::Organization  {
              system = os:metaGaia::System{}
            };

            when { ProjectMapping(p,os); }

            where{ OrganizationMapping(domain1,domain2); }
      }

      relation OrganizationMapping {

          varName : String;
          varGoal : String;

            checkonly                    domain                    source
domain1:metaRE4Gaia::functionalRefinementTree::FunctionalRefinementTre
e  {
             suborganizations            =                         so:
metaRE4Gaia::functionalRefinementTree::SubOrganization {
                          name = varName, description = varGoal
             }
            };

            enforce                      domain                    target
domain2:metaGaia::organizationModel::Organization  {
             suborganizations            =            sb           :
metaGaia::organizationModel::Suborganization {
              name = varName, goal = varGoal
             }
            };

      }


    /* Maps a Role Model */
    top relation RoleModelMapping{
          checkonly                      domain                    source
domain1:metaRE4Gaia::requirementsRoleModel::RequirementsRoleModel{
            project = p:metaGaia::Project{}
           };
```

```
            enforce                         domain                          target
domain2:metaGaia::roleModel::RoleModel{
              system = os:metaGaia::System{}
            };
             when{
              ProjectMapping(p,os);
             }
             where{
              RoleMapping(domain1,domain2);
             }
    }

    /* Maps a single role */
    relation RoleMapping {

        varName : String;
        varDescription : String;

            checkonly                       domain                          source
domain1:metaRE4Gaia::requirementsRoleModel::RequirementsRoleModel  {
             roles = rrm:metaRE4Gaia::functionalRefinementTree::Role{
                  name = varName,
                  description = varDescription
             }
            };

            enforce                         domain                          target
domain2:metaGaia::roleModel::RoleModel  {
             roles = rm:metaGaia::roleModel::Role{
               name = varName,
               description = varDescription
             }
            };
      }
      /* Protocol Mapping */
      top relation InteractionModelMapping{
            checkonly domain source domain1:metaRE4Gaia::Project{
            };

            enforce                         domain                          target
domain2:metaGaia::interactionModel::InteractionModel{
             system = s : metaGaia::System {}
            };

            when{
             ProjectMapping(domain1,s);
            }
      }

      top relation ProtocolMapping {

          varName : String;
```

```
          varDescription : String;
          varInitiator : String;
          varPartner : String;

            checkonly                    domain                    source
domain1:metaRE4Gaia::activityDiagram::SendSignalAction {
             name = varName,
             functions              =              fun              :
metaRE4Gaia::functionalRefinementTree::Function   {   description   =
varDescription},
             target                                                  =
prtn:metaRE4Gaia::activityDiagram::AcceptEventAction{
              partition              =              pat              :
metaRE4Gaia::activityDiagram::ActivityPartition {
               role = nam : metaRE4Gaia::functionalRefinementTree::Role
{
                name = varPartner
               }
              }
             },
             partition                                               =
pa:metaGaia::activityDiagram::ActivityPartition{
              role                                                   =
rl:metaGaia::activityDiagram::functionalRefinementTree::Role{
               name = varInitiator
              },
              diagram                                                =
da:metaRE4Gaia::activityDiagram::ActivityDiagram{
               suborganization                                       =
so:metaRE4Gaia::functionalRefinementTree::SubOrganization{
                frt                                                  =
fr:metaRE4Gaia::functionalRefinementTree::FunctionalRefinementTree{
                 project = p:metaRE4Gaia::Project{}
                }
               }
              }
             }
            };

            enforce                    domain                    target
domain2:metaGaia::interactionModel::Protocol  {
              name = varName,
              description = varDescription,
              initiator = irl:metaGaia::roleModel::Role{
               name = varInitiator
              },
               partner = prtnr : metaGaia::roleModel::Role {
                name = varPartner
               },
              interactionModel                                       =
im:metaGaia::interactionModel::InteractionModel{
               system = s:metaGaia::System{}
```

```
            }
          };

          when{
           ProjectMapping(p,s);
           InteractionModelMapping(p,im);
          }
    }

    /* Activity */
    top relation ActivityMapping {

          varName : String;
          varDescription : String;
          varRoleName : String;

          checkonly domain source
domain1:metaRE4Gaia::activityDiagram::Activity  {
          name = varName,
          functions = fun :
metaRE4Gaia::functionalRefinementTree::Function {description =
varDescription},
           partition =
par:metaRE4Gaia::activityDiagram::ActivityPartition{
            diagram = dia:
metaRE4Gaia::activityDiagram::ActivityDiagram {
             suborganization =
so:metaRE4Gaia::functionalRefinementTree::SubOrganization{
              frt =
fr:metaRE4Gaia::functionalRefinementTree::FunctionalRefinementTree{
               project = p:metaRE4Gaia::Project{
                models =
rmm:metaRE4Gaia::requirementsRoleModel::RequirementsRoleModel{}
               }
              }
             }
            },
            role = rl:metaRE4Gaia::functionalRefinementTree::Role{
             name = varRoleName
            }
           }
          };

          enforce domain target domain2:metaGaia::roleModel::Activity
{
           name = varName,
           description = varDescription,
           role = rol:metaGaia::roleModel::Role{
            name = varRoleName,
            roleModel = rlm:metaGaia::roleModel::RoleModel{
             system = s:metaGaia::System{}
            }
```

```
            }
           };

           when{
            ProjectMapping(p,s);
            RoleModelMapping(rmm,rlm);
           }

      }

      /* Map a Comment to a Responsability Rule */
      /* Liveness */
      top relation CommentToLivenessResponsability {

            varDescription : String;
            varRole : String;
            /* PERMISO */

            checkonly                  domain                  source
domain1:metaRE4Gaia::activityDiagram::Comment {
             description = varDescription,
             partition             =             par             :
metaRE4Gaia::activityDiagram::ActivityPartition {
                  role             =             nam             :
metaRE4Gaia::functionalRefinementTree::Role {
                        name = varRole
                  }
             },
             type = metaRE4Gaia::enumerates::RuleType::Liveness
           };

            enforce                    domain                  target
domain2:metaGaia::roleModel::LivenessResponsibility  {
                  description = varDescription,
                  role = be : metaGaia::roleModel::Role {
                        name = varRole
                  }
           };
      }

      /* Security */
      top relation CommentToSecurityResponsability {

            varDescription : String;
            varRole : String;
            /* PERMISO */

            checkonly                  domain                  source
domain1:metaRE4Gaia::activityDiagram::Comment {
             description = varDescription,
             partition             =             par             :
metaRE4Gaia::activityDiagram::ActivityPartition {
```

```
                role                =                nam                :
metaRE4Gaia::functionalRefinementTree::Role {
                        name = varRole
                }
             },
             type = metaRE4Gaia::enumerates::RuleType::Safety
            };

            enforce                    domain                    target
domain2:metaGaia::roleModel::SecurityResponsibility  {
                description = varDescription,
                role = be : metaGaia::roleModel::Role {
                        name = varRole
                }
            };
      }


      /* Environment */
      top relation EnvironmentalMapping {

            checkonly                    domain                    source
domain1:metaRE4Gaia::environmentalModel::EnvironmentalModel  {
              project = p:metaRE4Gaia::Project{}
            };

            enforce                    domain                    target
domain2:metaGaia::environment::Environment  {
              system = sy:metaGaia::System{}
            };

            when {
             ProjectMapping(p,sy);
            }

            where {
             ResourceMapping(domain1,domain2);
            }

      }

      /* Resource Mapping */
      relation ResourceMapping {

          varName : String; varDescription : String;

            checkonly                    domain                    source
domain1:metaRE4Gaia::environmentalModel::EnvironmentalModel  {
             resources = r : metaRE4Gaia::environmentalModel::Resource
{
              name = varName, description = varDescription
             }
```

```
            };

            enforce                    domain                    target
domain2:metaGaia::environment::Environment  {
             resources = rs : metaGaia::environment::Resource {
              name = varName, description = varDescription
             }
            };

      }

      /* PERMISSIONS */
      /* READ */
      top relation PermissionMappingREAD {

            varName : String;
            varResource : String;
            /* PERMISO */

            checkonly                  domain                    source
domain1:metaRE4Gaia::environmentalModel::Permission {
             role = rl : metaRE4Gaia::functionalRefinementTree::Role {
              name = varName
             },
             resource = rse:metaRE4Gaia::environmentalModel::Resource{
              name = varResource
             },
             type = metaRE4Gaia::enumerates::PermissionType::Read
            };

            enforce                    domain                    target
domain2:metaGaia::roleModel::Permission  {
             role = rol : metaGaia::roleModel::Role {
              name = varName
             },
             resource = res:metaGaia::environment::Resource{
              name = varResource
             },
             name = varResource + '-' + 'Read',
             type = metaGaia::enumerates::PermissionKind::Read
            };
      }

      /* write */
      top relation PermissionMappingWRITE {

            varName : String;
            varResource : String;
            /* PERMISO */

            checkonly                  domain                    source
domain1:metaRE4Gaia::environmentalModel::Permission {
```

```
            role = rl : metaRE4Gaia::functionalRefinementTree::Role {
             name = varName
            },
            resource = rse:metaRE4Gaia::environmentalModel::Resource{
             name = varResource
            },
            type = metaRE4Gaia::enumerates::PermissionType::Modify
           };

          enforce                    domain                    target
domain2:metaGaia::roleModel::Permission  {
            role = rol : metaGaia::roleModel::Role {
             name = varName
            },
            resource = res:metaGaia::environment::Resource{
             name = varResource
            },
            name = varResource + '-' + 'Modify',
            type = metaGaia::enumerates::PermissionKind::Modify
           };
      }

      /* consume */
      top relation PermissionMappingCONSUME {

            varName : String;
            varResource : String;
            /* PERMISO */

          checkonly                  domain                    source
domain1:metaRE4Gaia::environmentalModel::Permission {
            role = rl : metaRE4Gaia::functionalRefinementTree::Role {
             name = varName
            },
            resource = rse:metaRE4Gaia::environmentalModel::Resource{
             name = varResource
            },
            type = metaRE4Gaia::enumerates::PermissionType::Consume
           };

          enforce                    domain                    target
domain2:metaGaia::roleModel::Permission  {
            role = rol : metaGaia::roleModel::Role {
             name = varName
            },
            resource = res:metaGaia::environment::Resource{
             name = varResource
            },
            name = varResource + '-' + 'Consume',
            type = metaGaia::enumerates::PermissionKind::Consume
           };
      }
```

```
      /* Rule Model */
      top relation RuleModelMapping{
           checkonly                    domain                    source
domain1:metaRE4Gaia::organizationalRulesModel::OrganizationalRulesMode
l{
            project = p:metaGaia::Project{}
           };
           enforce                     domain                    target
domain2:metaGaia::organizationRules::OrganizationalRulesModel{
             system = os:metaGaia::System{}
           };
            when{
                  ProjectMapping(p,os);
            }
            where{
                  LivenessOrganizationalRule(domain1,domain2);
                  SafetyOrganizationalRule(domain1,domain2);
            }
    }

    /* Liveness Organizational Rule */
    relation LivenessOrganizationalRule{
      varDescription : String;

      checkonly                     domain                    source
domain1:metaRE4Gaia::organizationalRulesModel::OrganizationalRulesMode
l{
            rules = rl:metaRE4Gaia::organizationalRulesModel::Rule{
                  type = metaRE4Gaia::enumerates::RuleType::Liveness,
                  description = varDescription
            }
      };

      enforce                     domain                    target
domain2:metaGaia::organizationRules::OrganizationalRulesModel{
            rules = rul:metaGaia::organizationRules::LivenessRule{
                  description = varDescription
            }
      };
    }

    /* Safety Organizational Rule */
    relation SafetyOrganizationalRule{
      varDescription : String;

      checkonly                     domain                    source
domain1:metaRE4Gaia::organizationalRulesModel::OrganizationalRulesMode
l{
            rules = rl:metaRE4Gaia::organizationalRulesModel::Rule{
                  type = metaRE4Gaia::enumerates::RuleType::Safety,
                  description = varDescription
```

```
            }
      };

      enforce                    domain                    target
domain2:metaGaia::organizationRules::OrganizationalRulesModel{
            rules = rul:metaGaia::organizationRules::SafetyRule{
                  description = varDescription
            }
      };
    }
}
```

Bibliografía

American Association for Artificial Intelligence. (n.d.). *American Association for Artificial Intelligence*. Retrieved 11 01, 2009, from http://www.aaai.org/AITopics/index.html

Argente, E., Botti, V., & Julián, V. (DCAI 2009). Organizational-Oriented Methodological Guidelines for Designing Virtual Organizations. *International Symposium on Distributed Computing and Artificial Intelligence. 2*, pp. 154-162. Salamanca, Spain: LNCS Springer.

Bernon, C., Cossentino, M., & Pavón, J. (2005). An Overview of Current Trends in European AOSE Research. *Journal of Informatica 29 (4) , 29* (4), 379--390.

Beydeda, S., Book, M., & Gruhn, V. (2005). *Model-Driven Software Development.* New York, USA: Springer.

Blanes, D., Insfrán, E., & Abrahão, S. (2009). International Symposium on Distributed Computing and Artificial Intelligence. *DCAI 2009* (pp. 134-137). Salamanca, Spain: LNCS Springer.

Blanes, D., Insfrán, E., & Abrahão, S. (2009). Requirements Engineering in the Development of Multi-Agent Systems: A Systematic Review. *International Conference on Intelligent Data Engineering and Automated Learning* (pp. 510-517). Burgos, Spain: Springer.

Burmeister, B., Arnold, M., Copaciu, F., & Rimassa, G. (2008). *BDI-agents for agile goal-oriented business processes.* AAMAS (Industry Track) 2008: 37-44.

Castro, J., Kolp, M., & Mylopoulos, J. (2002). Towards requirements-driven information systems engineering: the Tropos project. *Information Systems, Volume 27* , 365-389.

Cernuzzi, L., Cossentino, M., & Zambonelli, F. (2005). Process Models for Agent-based Development. Engineering Applications of Artificial Intelligence 18(2), pp.

Cheng, B., & Atlee, J. (2007). *Research directions in Requirements Engineering.* Future of Software Engineering,. Volume , Issue , Page(s):285 - 303.

Davis, A. M., Dieste, Ó., Hickey, A. M., Juristo Juzgado, N., & Moreno, A. M. (2006). *Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review.* Minneapolis/St.Paul, Minnesota, USA: RE.

Dehlinger, J., & Lutz, R. (2005). *A product-line requirements approach to safe reuse in multi-agent systems.* ACM SIGSOFT Software Engineering Notes 30(4): 1-7.

DeLoach, S. A., Wood, M. F., & Sparkman, C. H. (2001). Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering 11(3)* , 231-258.

Fuentes-Fernández, R., Gómez-Sanz, J. J., & Pavón, J. *Requirements Elicitation and Analysis of Multiagent Systems Using Activity Theory.* EE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humas, Vol. 39, No. 2, March 2009.

Fuggetta, A. (2000). Software process: a roadmap. *International Conference on on Software Engineering, Future of Software Engineering Track* (pp. 25-34). Limerick, Ireland: ACM.

Genesereth, M. R., & Ketchpel, S. P. (1994). Software Agents. *Communications of the ACM (CACM), Volume 37* , 48-53.

Giret, A., Botti, V. J., & Valero, S. (2005). MAS Methodology for HMS. *Holonic and Multi-Agent Systems for Manufacturing, Second International Conference on Industrial Applications, of Holonic and Multi-Agent Systems, HoloMAS 2005* (pp. 39-49). Copenhagen, Denmark: Lecture Notes in Computer Science.

Gotel, O., & Finkelstein, A. (1994). *An Analysis of the Requirements Traceability Problem.* Colorado Springs, Colorado: Proceedings of the IEEE International Conference on Requirements Engineering (ICRE '94).

Henderson-Sellers, B., & Giorgini, P. (2005). *Agent-Oriented Methodologies.* Idea Group Inc.

Henderson–Sellers, B., & Gorton, I. (2002). *Agent-based Software Development Methodologies.* white paper, Summary of Workshop at the OOPSLA.

IEEE Std 610.12-1990. (1993). *IEEE Software Engineering Standard: Glossary of Software Engineering Terminology*. IEEE Computer Society Press.

IEEE Std 830-1998. (1998). IEEE Recommended Practice for Software Requirements Specifications. *IEEE Computer Society Press* .

IEEE Std. 610.12-1990. (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Computer Society Press* .

Iglesias, C. A., Garijo, M., Centeno-González, J., & Velasco, J. R. (1997). Analysis and Design of Multiagent Systems Using MAS-Common KADS. *Intelligent Agents IV, Agent Theories, Architectures, and Languages, 4th International Workshop, ATAL '97* (pp. 313-327). Providence, Rhode Island, USA: Lecture Notes in Computer Science.

Insfran, E. (2003). *A Requirements Engineering Approach for Object-Oriented Conceptual Modeling.* Valencia, Spain: Universidad Politécnica de Valencia.

Instituto de Investigación en Inteligencia Artificial (IIIA). (1998, 1 8). *The Fishmarket Project*. Retrieved 5 1, 2009, from http://www2.iiia.csic.es/Projects/fishmarket

Jacobson, I., & Jacobson, S. (2005). Reengineering your software engineering process. *Object Magazine* .

Juan, T., Pearce, A. R., & Sterling, L. (2002). ROADMAP: extending the gaia methodology for complex open systems. *AAMAS* (pp. 3-10). Bologna,Italy: ACM.

Kinny, D., & Georgeff, M. P. (1997). Modelling and Design of Multi-Agent Systems. *Intelligent Agents III, Agent Theories, Architectures, and Languages, ECAI '96 Workshop (ATAL)* (pp. 1-20). Budapest, Hungary: Lecture Notes in Computer Science.

Kitchenham, B. (2004). *Procedures for Performing Systematic Reviews.* Australia: Joint Technical Report Software Engineering Group, Keele University, United Kingdom and Empirical Software Engineering, National ICT Australia Ltd.

Lind, J. (2001). Iterative Software Engineering for Multiagent Systems, the MASSIVE Method.

Mendes, E. (2005). *A systematic review of Web engineering research.* Noosa Heads, Australia: International Symposium on Empirical Software Engineering.

Model Driven Architecture. (2009, 11 1). Retrieved 11 1, 2009, from http://www.omg.org/mda/

Napoli, C. D., Giordano, M., & Furnari, M. M. (1996). A PVM implementation of the Fishmarket. *IX International Symposium on Artificial Intelligence.* Cancun.

Nuseibeh, B., & Easterbrook, S. M. (2000). Requirements engineering: a roadmap. *ICSE 2000 - Future of SE Track: Limerick* (pp. 35-46). Limerick, Ireland: ACM.

Object Management Group. (2009, 11 1). Retrieved 11 1, 2009, from http://www.omg.org/

Object Management Group. (2008, abril). *Documents associated with Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1.0* . Retrieved noviembre 3, 2009, from http://www.omg.org/spec/QVT/1.0/

Object Management Group. (2005, 01 06). *Software Process Engineering Meta-Model (SPEM), version 1.1*. Retrieved 11 03, 2009, from http://www.omg.org/cgi-bin/doc?formal/05-01-06.pdf

Paraense, A., Gudwin, R., & Gonçalves, R. (2007). *Brainmerge: a Semiotic-Oriented Software Development Process for Intelligence Augmentation Systems.* KIMAS.

Pavón, J., & Gomez, J. (2003). Agent Oriented Software Engineering with INGENIAS. *CEEMAS* (pp. 394-403). Prague, Czech Republic: Springer.

Penserini, L., Perini, A., Susi, A., & Mylopoulos, J. (2007). High Variability Design for Software Agents: Extending Tropos. *ACM Trans. Autonom. Adapt. Syst. 2, 4, Article 16.*

Ranjan, P., & Misra, A. (2006). *A hybrid model for agent based system requirements analysis.* ACM SIGSOFT Software Engineering Notes 31(3): 1-7.

Ricordel, P. M. (2001). *Programmation Orientée Multi-Agents* . Institut National Polytechnique de Grenoble.

Rodríguez, J. A., Noriega, P., Sierra, C., & Padget, J. (1997). FM96.5 A Java-based Electronic Auction House. *PAAM.*

Silva, C., Tedesco, P., Castro, J., & Pinto, R. (2004). *Comparing Agent-Oriented Methodologies Using the NFR Approach.* Edinburgh, Scotland, UK: 26th International Conference on Software Engineering.

Sommerville, I. (2002). *Ingeniería de Software.* Pearson Educación.

Sudeikat, J., Braubach, L., Pokahr, A., & Lamersdorf, W. (2004). *Evaluation of Agent-Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform.* New York, USA: AOSE.

Tennenhouse, D. L. (2000). Proactive Computing. *Communications of the ACM (CACM) , 43*, 43-50.

The Eclipse Foundation. (2009, 11 1). *Eclipse Modeling Framework Project*. Retrieved 11 21, 2009, from http://www.eclipse.org/modeling/emf

The Eclipse Foundation. (2009, 11 1). *Graphical Modelling Framework (GMF)*. Retrieved 11 1, 2009, from http://www.eclipse.org/gmf/

Wagner, G. (2003). The Agent-Object-Relationship metamodel: towards a unified view of state and behavior. *Information Systems, Volume 28, Number 5* , 475-504.

Wilmann, D., & Sterling, L. (2005). *Guiding agent-oriented requirements elicitation: HOMER.* QSIC.

Wooldridge, M. (2004). *An Introduction to Multi-Agent Systems.* John Wiley & Sons.

Wooldridge, M. J., & Jennings, N. R. (1994). Agent Theories, Architectures, and Languages: A Survey. *ECAI-94 Workshop on Agent Theories, Architectures, and Languages* (pp. 1-39). Amsterdam, The Netherlands: Springer-Verlag.

Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems 3(3)* , 285-312.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing Multiagent Systems:The Gaia Methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)* , 317-370.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2001). Organizational rules as an abstraction for the analysis and design of multi-agent systems. *Int. J. Softw. Knowl. Eng. 11, 3* , 303–328.

Zambonelli, F., Jennings, N. R., Omicini, A., & Wooldridge, M. (2001). *Agent-Oriented Software Engineering for Internet Applications.* Springer.

Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys, Volume 27* , 315-321.