PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

DESARROLLO DE HERRAMIENTA DE MODELADO PARA METODOLOGÍA PASSI

DEYANIRA ANDREA SILVA URTUBIA

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

DICIEMBRE 2008

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

DESARROLLO DE HERRAMIENTA DE MODELADO PARA METODOLOGÍA PASSI

DEYANIRA ANDREA SILVA URTUBIA

Profesor Guía: **Claudio Cubillos Figueroa**

Profesor Co-referente: **Nibaldo Rodriguez Agurto**

Carrera: **Ingeniería de Ejecución en Informática**

DICIEMBRE 2008

A mis dulces princesitas, que con sus sonrisas me llenan de vida cada día. A mi esposo, por su amistad y amor incondicional. A mi papá, que desde el cielo me acompaña en cada paso que doy. Y sobre todo austed mamá, que ha sido el pilar de mi vida y sin su apoyo nada de esto sería posible.

Deya

Resumen

Las nuevas problemáticas de hoy en día han creado la necesidad de buscar nuevas metodologías, donde las aplicaciones complejas que poseen autonomía se han convertido en una solución apetecible, como lo son los sistemas multiagente. Para modelar estos sistemas multiagente existen varias metodologías que ofrecen un modelado completo pero cada uno desde distintos puntos de vista, como lo son MaSE, Gaia y PASSI. PASSI ofrece un modelado paso a paso desde los requerimientos a código utilizando diagramas UML que logran un modelado detallado y completo del sistema multiagente. En esta tesis se presenta la creación de una herramienta que permite modelar un sistema multiagente de acuerdo a las directrices planteadas por la metodología PASSI y que corrija los inconvenientes que presenta el PASSIToolKit de IBM.

Esta herramienta se creó basándose en Violet, un editor de UML de código abierto que permite modificar su código para crear nuevas utilidades. Para crear y modificar el código de Violet se utilizó el IDE Eclipse. Tomando este editor como punto de partida se crearon clases 'maestras' que permiten la comunicación entre los diversos diagramas que se van generando a medida que avanza el proceso de modelado.

Finalmente se logró que la herramienta creada fuera capaz de realizar los modelos planteados por PASSI hasta la tercera fase, esto es, hasta el Modelo de Implementación de Agentes. Queda para un trabajo futuro la depuración de la herramienta creada y completar los modelos inconclusos.

Palabras Claves: Sistemas Multiagente, metodologías, PASSI, herramienta de modelado, Violet.

Abstract

Today's new problematics have created the need of new technologies where autonomous complex applications have become a desire solution such as multiagent systems. For modeling these multiagent systems several methodologies exist that offer a complete modeling but each one from its own point of view like MaSE, Gaia and PASSI. PASSI offers a step by step modeling from requirements-to-code using UML diagrams that accomplish a complete and detailed modeling of the multiagent system. This work presents the creation of a tool that allows a multiagent system modeling according to the guidelines proposed by the PASSI methodology and that can be able to correct the inconveniences of the IBM PASSI ToolKit.

This tool was created over Violet, an open-code UML editor that allows the modifications of it own code for the creation of new utilities. To make these changes to the violet code the Eclipse IDE was used. Taking this editor as starting point, it was necessary to create some master classes to allow the communication between diverse diagrams, which are generated during the modeling process.

Finally the created tool is capable of managing any model from PASSI until the third stage, which means, until the agent implementation model. For future developments the next level can be the depuration of the created tool and to complete the unconcluded models.

Key Words: Multiagent systems, AOSE methodologies, PASSI, modeling tool, Violet.

Índice

Lista de Figuras

Lista de Tablas

1 Diseño de la Aplicación

Debido al aumento de la necesidad de crear aplicaciones complejas con cierta autonomía, también ha aumentado el uso de sistemas multiagente para resolver este tipo de problemática [3]. El paradigma de agentes y sistemas multiagente constituye actualmente un área de creciente interés dentro de la Inteligencia Artificial, entre otras razones, por ser aplicable a la resolución de problemas complejos no resueltos de manera satisfactoria mediante técnicas clásicas. Numerosas aplicaciones basadas en este nuevo paradigma vienen ya siendo empleadas en infinidad de áreas, tales como control de procesos, procesos de producción, control de trafico aéreo, aplicaciones comerciales, gestión de información, comercio electrónico, aplicaciones medicas, juegos, etc.

El proceso de diseño de un sistema multiagente no implica solo el modelar reemplazando un agente por un objeto, sino implica también capturar el dominio de la ontología [7], representar su interacción con otros agentes y entregarle la habilidad de tener comportamientos inteligentes. Es por esto que se puede decir que un sistema multiagente difiere de un sistema no basado en agentes en que, los agentes intentan ser unidades autónomas con funcionalidades inteligentes. Por consecuencia, los métodos de ingeniería de software basados en agentes deben complementar los diseños de actividades estándar y sus representaciones con modelos de sociedad de agentes.

En la literatura existente sobre el diseño y modelado de sistemas multiagente se pueden encontrar variados trabajos científicos con diferentes aproximaciones y desde distintos campos de investigación, como la Inteligencia Artificial, la Ingeniería de Software y, en algunos casos, hasta la Robótica. Cada una de estas áreas da énfasis a diferentes aspectos del proceso, sin embargo, la mayorías de ellas trabaja con los mismos elementos, aunque de diferente forma y con diferentes lenguajes o notaciones [8]. De acuerdo a esto se han propuesto diversas metodologías (MASE, Gaia, CASSIOPEIA) sin embargo, es PASSI quien coordina de mejor forma el análisis de las entradas y de las actividades a ser ejecutadas, así como la automatización de la mayoría de los diferentes pasos del proceso (o lo más cercano a esto, entregando al diseñador un gran soporte). PASSI [5] (“a Process for Agent Societies Specification and Implementation” o “pasos” en italiano) es una metodología paso a paso de requerimiento-a-código para el diseño y desarrollo de sociedades multiagente, integrando modelos de diseño y conceptos de ingeniería de software orientado a objetos y MAS utilizando el Lenguaje de Modelado Unificado (UML).

1.1 Problemática

En los últimos años se ha ido incrementando la atención en los sistemas multiagente (MAS), lo que ha provocado una necesidad de los ingenieros de realizar una ingeniería de software de gran calidad para la realización de sus diseños. Debido a esto han aparecido diferentes aproximaciones que tratan de presentar una metodología apropiada para el desarrollo de sistemas multiagente, siendo PASSI una de las mas importantes y mas usadas en el mercado, ya que es la que mas se acerca al concepto de alta fidelidad por la calidad de sus modelos.

Debido a esto es importante y necesario contar con herramientas que faciliten y automatizan el proceso de modelado de PASSI, de manera que los desarrolladores concentren sus esfuerzos en el diseño en si de los sistemas multiagente y no en el como reflejar el modelado propiamente tal.

Para apoyar el diseño de la metodología, existe hasta el momento una única herramienta CASE [6]; se trata de un plug-in para Racional Rose de IBM llamado PTK (Passi ToolKit). Esta herramienta permite modelar con PASSI generando los diferentes diagramas que requieren cada uno de los modelos de esta metodología, así como la autogeneración de código a partir de dichos diagramas. . El gran inconveniente de esta herramienta es el hecho de tener que usarse si o si en una aplicación que requiere ser comprada por lo que no es accesible a todos.

Pese a que PASSI tiene una gran aceptación y su utilización se ha extendido radicalmente, los diseñadores no cuentan con herramientas alternativas variadas para realizar el modelado con esta metodología. Más aun, la única herramienta existente, PTK, solo se puede utilizar con Racional, que es un software comercial, con fallas y de un coste elevado. La otra gran dificultad con la que se deben enfrentar los desarrolladores es el hecho de no contar con suficiente información acerca del funcionamiento de PTK.

Debido a esto se hace necesario contar con una herramienta que pueda ser utilizada de forma libre (código abierto) dentro de una aplicación también libre, que cuente con las características fuertes de PTK y que, a su vez, mejore las debilidades que éste presente.

1.2 Definición de Objetivos

Para el desarrollo del presente proyecto se han definido los objetivos que se presentan a continuación:

1.2.1 Objetivo Principal

Desarrollar un plugin para el IDE Eclipse que permita el modelado de la metodología PASSI basado en el plugin VIOLET.

1.2.2 Objetivo Secundario

- Recopilar y analizar documentación sobre la metodología de modelado PASSI
- Analizar soluciones existentes de modelado para PASSI y su estado del arte
- Desarrollar una herramienta que permita mediante interfaces graficas modelar sistemas multiagente con la metodología PASSI hasta su tercer modelo correspondiente al Modelo de Implementación de Agentes.
- Realizar pruebas de validación a la herramienta de modelado desarrollada.

1.3 Metodología

Al igual que en los paradigmas, existen distintos tipos de metodologías de desarrollo, según la naturaleza del proyecto. Estas pueden variar desde simples exposiciones narrativas hasta métodos rigurosamente formales. La correcta elección de una metodología es una de las decisiones más relevantes que debe tomar el analista, para lograr de esta forma un producto de software, que cumpla con todas las expectativas que de él se esperan

Para el desarrollo del presente proyecto se utilizarán dos metodologías que permitirán definir y desarrollar tanto la búsqueda de información como el enfoque que se le dará al proyecto.

Para la obtención de información fidedigna se utilizara la Metodología Exploratoria [1], esta permitirá recabar toda la información necesaria sobre el estado del arte referente al proyecto, esto es, de los sistemas multiagente en si y de la problemática que trata de resolver el proyecto que se esta presentando.

Se ha decidido que la metodología de Análisis Orientado a Objeto será la utilizada en el desarrollo del proyecto, debido a que nos facilita la tarea de poder identificar los elementos que componen un sistema, y así reducir las distancias entre las actividades de análisis, porque trata los atributos y métodos de los elementos del sistema como un todo, y además permite al analista y al cliente poder tener una mejor comunicación ya que esta tiene herramientas de representación muy comprensibles para ambas partes.

El Análisis Orientado a Objetos [15] maneja métodos que permiten al ingeniero de software modelar un problema a través de la representación de objetos, atributos y operaciones como las componentes primarias del modelado. Una amplia variedad de métodos de análisis orientado a objetos han sido propuestos, pero todos poseen un conjunto de características posibles:

- Representación de clases o jerarquías de clases.
- Creación de modelos objeto-relación.
- Derivación de modelos objeto-comportamiento.

El análisis de sistemas orientados a objeto ocurre a muchos niveles diferentes de abstracción. Al nivel de negocios o empresarial, las técnicas asociadas con AOO pueden acoplarse con un enfoque de ingeniería de la información. Esta técnica se denomina normalmente análisis de dominio. Al nivel de aplicación, el modelo de objetos se centra en los requisitos específicos del cliente, pues estos afectan a la aplicación que se va a construir.

Los objetos modelan casi cualquier aspecto identificable del ámbito del problema: entidades externas, cosas, sucesos, papeles, unidades organizativas, lugares y estructuras, todas ellas pueden ser representadas como objetos.

Para el desarrollo de ésta se consideran fundamentalmente los siguientes pasos:

- Identificar los objetos considerando que los depósitos de datos que aparecen en los Diagramas de Flujo de Datos.
- Identificar las operaciones asociadas a cada objeto, para lo cual se realizan los procesos y se determina a qué objetos potenciales la asociación resulta más natural.

Para ello es necesario considerar que:

- Algunos candidatos a objetos no tienen operaciones que los acrediten como susceptibles de convertirse en objetos.
- Algunos procesos presentan correspondencia con más de un objeto, caso en el cual es necesario descomponer esos procesos.
- Habrá procesos sin candidatos a objetos. En este caso, se crea un objeto para cada uno de estos procesos.
- Representar objetos identificados, operaciones asociadas a entidades externas, y ajustar los diagramas de flujo de datos a las modificaciones introducidas.
- Definir las interfaces de los objetos.
- Realizar una evaluación final.

1.4 Paradigma

Para el desarrollo del proyecto, el paradigma elegido es, como se mencionó anteriormente, Proceso Unificado (UP). Se eligió este paradigma [11] debido a la necesidad de integrar múltiples facetas del desarrollo, esto es que sea capaz de:

- Proporcionar una guía ordenada de las tareas a realizar.
- Dirigir las tareas por separado de cada desarrollador y del equipo como un todo.
- Especifique los artefactos a desarrollarse.
- Ofrezca criterios de control y medición de los productos y actividades del proyecto.

1.4.1 Proceso Unificado

El proceso Unificado es un proceso de desarrollo de software, definido como el conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema de software. Sin embargo, es más que un simple proceso, es un marco de trabajo genérico que puede personalizarse para una gran variedad de sistemas de software, diferentes áreas de aplicación, tipos de organización, niveles de aptitud y tamaños de proyecto.

El proceso Unificado esta basado en componentes, lo que quiere decir que el sistema en construcción esta formado por componentes de software interconectados a través de interfaces bien definidas. Utiliza el Lenguaje Unificado de Modelado (UML) para preparar todos los esquemas de un sistema de software. Los aspectos relevantes del Proceso Unificado se resumen en tres características:

- Dirigido por casos de uso.
- Centrado en la arquitectura.
- Iterativo e incremental.

Dirigido por Casos de Uso

Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario una visión de lo que hará el sistema; los casos de uso modelan los requerimientos funcionales del sistema. El conjunto de ellos Modelo de Casos de Uso, en donde se describe la funcionalidad del sistema completo
El decir que esta dirigido por casos de uso significa que el proceso de desarrollo sigue un hilo, que avanza a través de una serie de flujos de trabajo que parten de los casos de uso. Estos casos de uso no se desarrollan aisladamente, sino que van a la par con la arquitectura del Sistema.

Centrado en la Arquitectura

El concepto de arquitectura incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura surge de las necesidades del cliente, como percibe el sistema el usuario, la plataforma en la que tiene que funcionar (hardware, sistema operativo, base de datos, protocolos de comunicación en red), los bloques de construcción reutilizables de que se disponen (Ej. marco para interfaces graficas de usuario), consideraciones de implantación, sistemas heredados, y requisitos no funcionales (Ej. Rendimiento o fiabilidad). Los casos de uso y la arquitectura están profundamente relacionados. Los casos de uso deben encajar en la arquitectura, y a su vez la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, actualmente y a futuro.

Iterativo e Incremental

El desarrollo de un producto software supone un gran esfuerzo que puede durar meses o años. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos. Cada mini proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos, al crecimiento del producto. En cada iteración los desarrolladores identifican y especifican los casos de uso relevantes y crean un diseño utilizando la arquitectura seleccionada como guía, para implementar dichos casos de uso. Si la iteración cumple sus objetivos, se continúa con la próxima. En caso contrario se deben revisar las decisiones previas y volver a iterar.

El poder controlar estas iteraciones entrega varios beneficios al proyecto que se esta desarrollando:

- La iteración controlada reduce el coste de riesgo a los costes de un sólo incremento.
- Reduce el riesgo de no cumplir con los compromisos adquiridos dentro de un período determinado.
- Acelera el ritmo total del esfuerzo del desarrollo, ya que se trabaja para conseguir metas claras a corto plazo.
- Los requerimientos del usuario, en vez de ser definidos en forma temprana, se van refinando a medida que se avanza en las iteraciones, haciendo más fácil la adaptación a los requisitos cambiantes.

1.4.1.1 Fases del Proceso Unificado

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida del sistema, donde cada uno de estos ciclos finaliza con una versión del sistema. Cada ciclo consta de cuatro fases: Inicio, Elaboración, Construcción y Transición.

Fase de Inicio

Durante la fase de inicio se desarrolla una descripción del producto final y se identifican y priorizan los riesgos mas importantes. El objetivo de esta fase es ayudar al equipo de proyecto a decidir cuales son los verdaderos objetivos del proyecto. Las iteraciones en esta etapa exploran diferentes soluciones posibles y diferentes arquitecturas posibles. Lo único que normalmente sobrevive a la fase de inicio es el incremento del conocimiento en el equipo.

Fase de Elaboración

Durante la fase de elaboración se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura. Las iteraciones de esta fase deben establecen una firme comprensión del problema a solucionar, detallar un plan para las siguientes iteraciones y eliminar los mayores riesgos. El objetivo final de esta fase es establecer una línea base de la arquitectura.

Fase de Construcción

Durante la fase de construcción se crea el producto y la línea base de la arquitectura crece hasta convertirse en el sistema completo. Al final de esta fase el producto contiene todos los casos de uso implementados. Esta fase finaliza con la capacidad Operativa Inicial. Esto se logra cuando el producto es lo suficientemente estable para ser usado, proporciona la gran mayoría de sus funcionalidades y esta listo para la fase de transición, aunque no libre de errores.

Fase de Transición

La fase de transición cubre el período durante el cual el producto se convierte en la versión beta. Las iteraciones en esta fase continúan agregando características al software; sin embargo, las características se agregan a un sistema que el usuario se encuentra utilizando activamente. Durante este periodo el equipo se encuentra ocupado fundamentalmente en corregir y extender la funcionalidad del sistema desarrollado en la fase anterior con la finalidad de obtener un sistema completo, con todas sus funcionalidades, estable y robusto. Esta fase concluye con el lanzamiento del producto una vez que se han cumplido con los objetivos fijados en la fase de Inicio.

Cada una de estas fases se subdivide en iteraciones, esto es que una fase completa se repite hasta llegar a un objetivo determinado. Cada una de estas iteraciones se desarrolla siguiendo un conjunto de cinco flujos de trabajo: *Requerimientos, Análisis, Diseño, Implementación y Pruebas*; donde cada uno estos flujos de trabajo genera una nueva versión del sistema, siendo estas un producto listo para la entrega. Constan de códigos fuentes, manuales y otros productos asociados. Sin embargo estos productos se deben ajustar no sólo a los requerimientos del cliente sino que también a la de la gente que va a trabajar con él.

El agrupamiento de actividades en flujos de trabajo es principalmente una ayuda para comprender el proyecto desde la visión tradicional en cascada. Estos flujos de trabajo están asociados a un conjunto de modelos que se desarrollan, y a su vez los modelos están compuestos por artefactos. Los artefactos más importantes a realizar son:

- Modelo de casos de uso
- Modelo de diseño
- Modelo de implementación
- Modelo de prueba

1.5 Alcances del Proyecto

Tomando en cuenta la envergadura y complejidad que presenta el desarrollar una herramienta de modelado que permita la correcta modelación de una metodología no tan conocida como PASSI, se debió decidir cual sería el punto de corte para este proyecto y que quedaría pendiente para un trabajo futuro. De acuerdo a esto se decidió que el presente proyecto abarcaría hasta la etapa de Modelo de Sociedad de Agentes por completo, que corresponde al segundo modelo de PASSI.

Al llegar a este punto se dará por finalizado el proyecto entregando toda la documentación existente, acompañada de un prototipo funcional que podrá modelar todas las etapas del proceso de modelado PASSI hasta el punto señalado anteriormente.

El sistema que se entregará será capaz de realizar los diversos diagramas que requiere PASSI, permitiendo avanzar consecutivamente por las fases que pide la metodología, sin embargo, será también capaz de realizar y reflejar las modificaciones que se realicen en pasos anteriores sin perder la consistencia del modelado. El sistema no entregara código, ya que no se llegará hasta esa etapa. El sistema permitirá guardar e imprimir los diagramas que se generen en formato de imagen respetando el diseño que se haya generado.

1.6 Plan de Trabajo

El desarrollo del proyecto se ha dividido según las fases propuestas por Proceso Unificado para el diseño del plan de trabajo. En la tabla que se presenta a continuación se pueden apreciar las iteraciones planteadas así como las fechas de inicio y termino de cada una de estas iteraciones.

Tabla 1.1: Plan de Trabajo

Fase	***N° Iteración***	***Objetivo a cumplir***	***Duración***
INICIO	1°	Completar la información requerida para el proyecto y seleccionar IDE de trabajo.	10-03-2008 al 29-03-2008
INICIO	2°	Definir la mayor cantidad de requerimientos del sistema y genera diagrama general de casos de uso.	31-03-2008 al 15-04-2008
ELABORACION	1°	Completar los requerimientos, detallar los principales casos de uso y corregir de ser necesario el diagrama general de casos de uso.	16-04-2008 al 30-04-2008
ELABORACION	2°	Finalizar el detalle de los casos de uso y comenzar definición de pruebas.	01-05-2008 al 15-05-2008
ELABORACION	3°	Definir la totalidad de casos de pruebas, tener un prototipo que describa el cuerpo básico del software y generar un plan detallado para las siguientes iteraciones.	16-05-2008 al 23-06-2008
CONSTRUCCION	1°	Determinar pruebas de usabilidad y obtener prototipo con funcionalidad de Modelo de Requerimientos del Sistema.	14-07-2008 al 04-08-2008
CONSTRUCCION	2°	Completar Caso de Prueba y obtener prototipo con funcionalidad de Modelo de Requerimientos del Sistema corregido	05-08-2008 al 15-09-2008

Tabla 1.1: Plan de Trabajo

Fase	*N° Iteración*	*Objetivo a cumplir*	*Duración*
CONSTRUCCION	3º	Obtener prototipo con funcionalidad de Modelo de Sociedad de Agentes y aplicar pruebas de usabilidad	22-09-2008 al 31-10-2008
CONSTRUCCION	4º	Obtener Manual de Usuario y realizar correcciones al prototipo anterior	03-11-2008 al 13-11-2008
TRANSICION	1°	Obtener producto Beta Realizar correcciones al prototipo	14-11-2008 al 27-11-2008
TRANSICION	2º	Entrega de la herramienta de modelado con su respectivo manual de usuario	28-11-2008

El documento que se está presentando se estructuró tomando como capítulos las partes mas relevantes del trabajo realizado. De acuerdo a este criterio el cuerpo del documento es el siguiente:

Capitulo 1: Diseño de la Aplicación. En esta primera sección se describe el proyecto explicando sus objetivos, alcances, plan de trabajo y análisis de factibilidad.

Capitulo 2: Sistemas Multiagente. En este capitulo se entrega una explicación de que son los sistemas multiagente, en que consisten y sus principales características. También se habla sobre la infraestructura de un agente y las diversas metodologías de desarrollo de sistemas multiagente.

Capitulo 3: PASSI. Al ser ésta la metodología escogida para el desarrollo del proyecto, se le ha dado un capitulo aparte a fin de profundizar en cada una de sus etapas y diagramas, así como la inter-relación entre dichos diagramas. También se habla en este capitulo sobre la herramienta PTK para modelado de PASSI.

Capitulo 4: IDE Eclipse. En este capitulo se habla sobre las virtudes de Eclipse, ya que es el IDE bajo el cual se desarrolla el proyecto, así como de su add-in Violet.

Capitulo 5: Desarrollo del Proyecto. El detalle del trabajo realizado y los avances del proyecto se encuentran en este capitulo, donde dicho trabajo esta subdividido según las fases que propone el Proceso Unificado.

Capitulo 6: Conclusiones. En este capitulo se encuentran las conclusiones obtenidas y el trabajo pendiente para el resto del proyecto.

Capitulo 7: Referencias. Aquí se encuentran las referencias utilizadas para la confección del presente documento.

ANEXO A: Plan de Pruebas. Se entrega el diseño del Plan de Pruebas.

ANEXO B: Manual de Usuario. Se presenta una guía rápida para la utilización de la aplicación por parte del usuario.

1.6.1 Carta Gantt

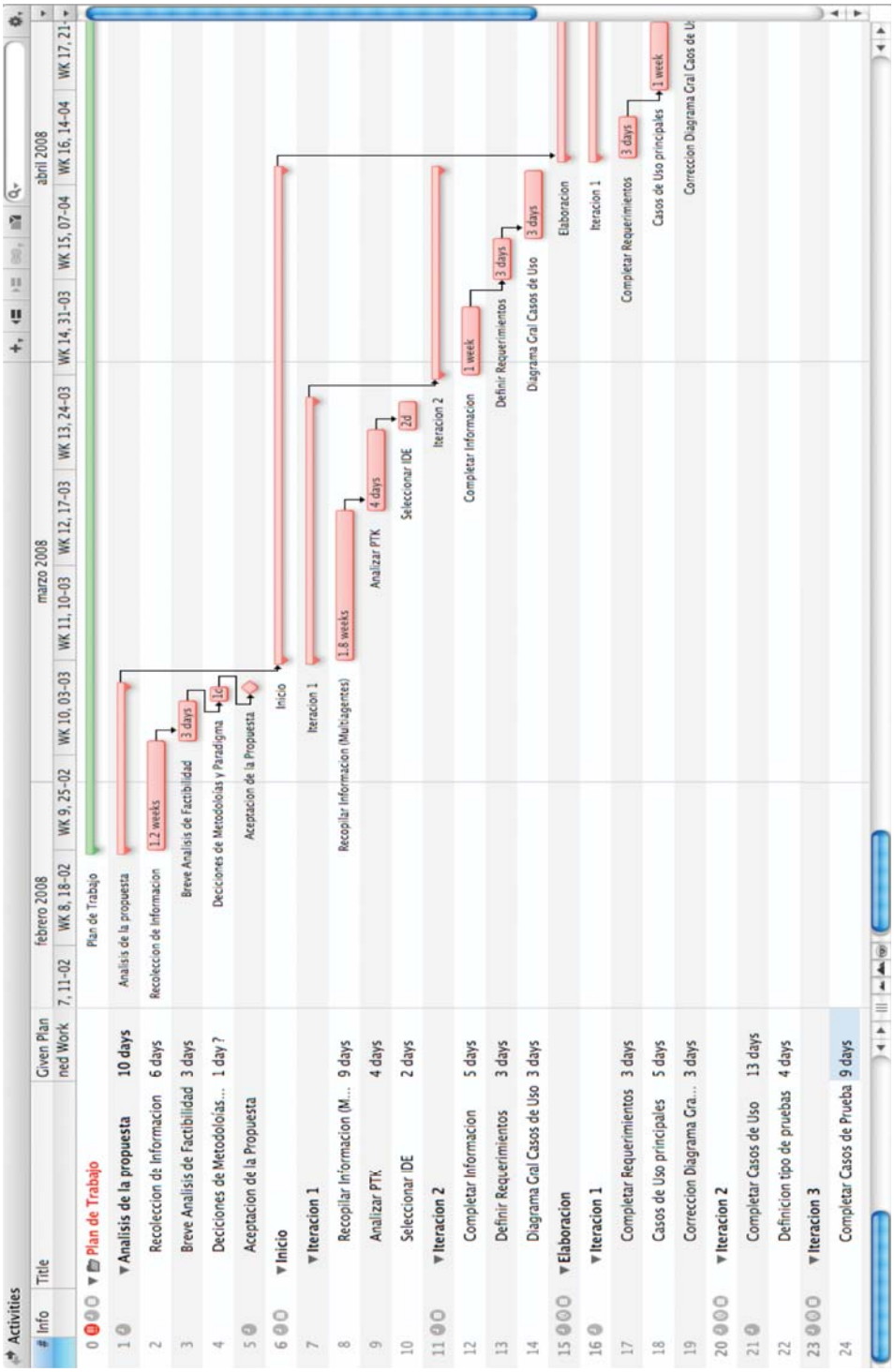

Figura 1.1: Carta Gantt

1.7 Gestión de Riesgos

Al comenzar con el análisis del proyecto, se debió realizar la gestión de riesgos a fin de determinar cuales serian los mayores riesgos asociados al proyecto, sus consecuencias en caso de presentarse y cuales serian los planes de mitigación asociados a dichos riesgos. La tabla 1.2 muestra los principales riesgos identificados en esta etapa.

Tabla 1.2: Gestión De Riesgos

Riesgo	Estrategia	Probabilidad y Efecto	Mitigación
Sub-estimación del tamaño del proyecto.	*Realizar un análisis exhaustivo y específico de los requerimientos y componentes del sistema.*	*25% Moderado*	*Realizar una nueva iteración a fin de detallar los requerimientos y ver los alcances del proyecto*
Falta de comprensión de los requerimientos por parte de los desarrolladores	*Listar los requerimientos con la mínima abstracción posible para evitar ambigüedades.*	*10 % Muy Bajo*	*Re-analizar los requerimientos a fin de lograr su completa comprensión.*
Sub-estimación del tiempo estimado.	*Realizar una planificación acuciosa y real.*	*30% Moderado*	*Revisar la planificación y realizar las modificaciones necesarias para acotar los tiempos.*
Poca experiencia con las tecnologías a utilizar y metodologías a desarrollar	*Planificar una curva de aprendizaje donde se estudie la tecnología y la metodología a desarrollar.*	*35% Moderado*	*Asumir el coste de tiempo que implica un nuevo periodo de aprendizaje*
Poca aceptación por parte del usuario a emplear la herramienta desarrollada.	*Implementar interfaces sencillas e intuitivas, así como diseñar manuales de usuario que entreguen información clara. Aplicar pruebes de usabilidad.*	*20% Bajo*	*Aplicar nueves pruebas de usabilidad y realizar los cambios correspondientes.*
Modificación de los requerimientos.	*Realizar un análisis minucioso a fin de identificar todos los requerimientos.*	*35% Moderado*	*Evaluar la posibilidad de cambios en el sistema.*

1.8 Estudio de Factibilidad

La finalidad del estudio de factibilidad es determinar si el proyecto que se está evaluando es o no viable, para esto se toman en cuéntalos diversos factores:

Análisis de Factibilidad Operacional

Se considera que la realización de esta aplicación constituirá un aporte a la comunidad de código abierto. Como es sabido, cada desarrollo libre se alimenta de un software libre existente y éste, a su vez, sirve como base para el siguiente desarrollo. Este método hace que el proceso sea mucho más dinámico y flexible, ahorrando costes temporales en análisis y gran parte del desarrollo. Este proyecto es un buen ejemplo de reutilización de desarrollos existentes. En este sentido, realizando el proyecto de esta forma, se garantiza de cierto modo la continuidad en esta línea de trabajo. Además, considerando el extendido uso de Eclipse, especialmente en los ámbitos profesionales y académicos, la creación de este plug-in, será bien recibido y considerado una valiosa herramienta por la gran comunidad detrás del proyecto eclipse.

Análisis de Factibilidad Técnica

La factibilidad técnica consiste en determinar si el sistema propuesto es viable desde el punto de vista de los recursos informáticos con los que cuenta la empresa, tanto computacionales como de comunicación, incluyendo los conocimientos técnicos respectivos, disponibles en la organización. En el caso de este proyecto no se cuenta con empresa alguna y los recursos con los que se cuentan son los que poseen los desarrolladores de forma personal.
Desde el punto de vista de los conocimientos de los desarrolladores, se considera que existen las condiciones necesarias para llevar acabo este proyecto ya que se cuentan con los conocimientos necesarios para desarrollar la aplicación, estos son:

- Conocimientos del modelado en PASSI (http://mozart.csai.unipa.it/passi/)
- Conocimientos de programación en lenguaje Java
- Experiencia previa con el IDE de Java a utilizar (Eclipse)

También se cuentan con los requerimientos recomendados para el desarrollo de la aplicación los que se detallan en la tabla 1.3.

Tabla 1.3: Requerimientos de Software

Requerimientos de Software	
Sistema Operativo	KUBUNTU, Windows XP Professional
Procesador de Texto	Open office Writer 2.4.0, Microsoft Word 2003
Navegador	Mozilla Firefox , o superiores
Correo Electrónico	Mozilla Thunderbird
Planilla de Cálculo	Openoffice Calc 2.4.0, Microsoft Excel 2003

Tabla 1.3: Requerimientos de Software

Requerimientos de Software	
Herramienta de desarrollo	Eclipse for RCP/PLUGIN Developers Europa
Presentación	Openoffice Impress 2.4.0, Microsoft Power Point 2003
Requerimientos de Hardware	
CPU	ATHLON XP, Duron 1000
Disco duro	Mínimo 40 GB
Memoria RAM	1024 MB
Tarjeta Madre	Compatible con CPU
Unidad óptica	CD/DVD-ROM
Accesorios	Teclado, mouse

Análisis de Factibilidad Legal

En cuanto al punto de vista legal, al basar todo el trabajo a realizarse en el campo del software`s libres, no existen impedimentos.
Se ha considerado conferir a este proyecto, una vez terminado, una licencia no comercial *Creative Commons*, lo que implica que el material original y los trabajos derivados pueden ser distribuidos, copiados y exhibidos mientras su uso no sea comercial.

Análisis de Factibilidad Económica

El objetivo de la factibilidad económica es evitar que un proyecto de software sea inviable en etapas críticas del proyecto por razones económicas, entrega los costos del proyecto, el impacto del sistema en la empresa y los costos de los recursos que se utilizaran en el desarrollo del sistema. De este modo el estudio de factibilidad económica sirve al propósito de tomar una decisión correcta en términos de relación costo/beneficio en relación directa con el proyecto.
En el caso del presente proyecto, al no contar con empresa alguna que corra con estos gastos, estos serán asumidos por los desarrolladores. Dichos costos se detallan en la tabla 1.4.

Tabla 1.4: Factibilidad Económica

Gastos Operacionales		
Implementos/ Servicios	Costo presupuestado	Costo Real
3 resma de papel tamaño carta	$ 6.000	$ 6.000
1 cartucho de impresión tinta negra	$ 18.500	$ 18.500
1 cartucho de impresión tinta color	$ 6.250	$ 6.250
5 anillados	$ 3.000	$ 3.000
Movilización para reuniones	$ 30.000	$ 30.000
Teléfono (Celular y Particular)	$ 15.000	$ 15.000
Internet	$ 115.200	$ 115.200
Subtotal	$ 273.600	$ 273.600
Gastos en Equipos		
2 Computadores	$ 800.000	$ 0 (equipo personal)
1 Impresora	$ 30.000	$ 0 (equipo personal)
2 Equipos Celulares	$ 56.000	$ 0 (equipo personal)
Sub Total	$ 886.000	$ 0
Gastos en Licencias		
Se utilizarán software libre	$ 0	$ 0
Sub Total	$0	$ 0
TOTAL	$1.159.600	$273.600

Considerando todos los puntos desarrollados anteriormente y que los gastos involucrados serán asumidos por los integrantes, se considera que el proyecto cuenta con la factibilidad necesaria para ser llevado a cabo.

2 Sistemas Multiagente

2.1 Agentes

Dentro de la literatura se pueden encontrar diversas definiciones de agente, esto se debe a que cada definición se apega al área en que se emplea el término. [5] Debido a esto no existe una definición formal universal de agente aceptada en el mundo científico. Sin embargo hay dos definiciones que representan de mejor manera lo que es un agente:

Según la definición de Russell un agente es *"Toda entidad que de forma autónoma perciba su entorno (real o simulado) mediante sensores y que actúa en el mismo mediante efectores"*..

Wooldridge explica un agente como *"un sistema informático que se sitúa en un cierto ambiente y que es capaz de realizar acciones flexibles y autónomas para lograr sus objetivos de diseño"* [16]

Ahora bien, a estas definiciones de agente se le debe agregar un adjetivo: "inteligente". Ahora "un agente inteligente realiza continuamente tres funciones: percepción de las condiciones dinámicas de su entorno; actúa de acuerdo a las condiciones de su entorno; razona para interpretar percepciones, resolver problemas, dibujar inferencias y determinar acciones" [2]

De acuerdo a estas definiciones se puede decir que un agente tiene tres características principales: ubicación en un ambiente, comportamiento flexible y autonomía.

Ubicación en un ambiente

El comportamiento del agente esta fuertemente relacionado al ambiente en donde se encuentra embebido. El agente percibe directamente el ambiente mediante sus sensores y actúa mediante sus efectores modificando el ambiente y sus percepciones futuras.

Comportamiento flexible

Debe ser flexible para cumplir con sus objetivos de diseño, donde ser flexible significa:

- *Reactivo*: el agente percibe su ambiente y responde en un tiempo adecuado a los cambios que se producen en él.
- *Pro-activo*: el agente no sólo actúa en respuesta al ambiente sino que puede tomar la iniciativa (comportamiento dirigido por el objetivo).
- *Social*: capacidad para interactuar, cuando es apropiado, con otros agentes artificiales o humanos.

Autónomo

- El agente debe ser capaz de actuar sin la intervención directa de humanos (u otros agentes).
- Tiene control sobre su propio estado interno (símil a objetos)
- Tiene control sobre su propias acciones (no existe una relación master/slave)
- Puede, si es necesario, modificar su comportamiento en base a la experiencia (aprender).

Otros de los atributos más comunes de agentes son:

- *Movilidad*: la capacidad de emigrar de una manera autónoma por la red.
- *Veracidad*: un agente no comunicará con conocimiento información falsa.
- *Benevolencia*: los agentes no tienen metas que están en conflicto, y que cada agente por lo tanto intentará siempre hacer lo que se le pide.
- *Racionalidad*: el agente actuará para alcanzar sus metas, y no actuará en modo tal de evitar el alcance de sus metas.
- *Adaptabilidad*: la capacidad de aprender y de mejorar con experiencia.

Se pueden definir dos nociones de agentes: Noción Fuerte y Noción Débil. La noción débil define al agente como una entidad capaz de intercambiar mensajes usando un lenguaje de comunicación de agentes y propone las características de autonomía, habilidad social, reactividad y pro-actividad para un agente. Por otra parte, la noción fuerte propone al agente como una entidad cuyo estado es visto como un conjunto de componentes mentales (capacidades, creencias, elecciones y acuerdos) y agrega a las características de noción débil las de movilidad, veracidad, benevolencia y racionalidad.

2.2 Sistemas Multiagente

Para definir los sistemas multiagente o MAS se utilizara la definición de Weiss: “Un sistema multiagente es una red de problem solvers que trabajan conjuntamente para encontrar respuestas a problemas que van más allá de las capacidades o conocimiento individuales de cada entidad.”; [7] dentro de esta definición se podría comparar un MAS con una organización.

Un MAS proporciona una serie de beneficios al trabajar los agentes en conjunto:

- *Velocidad y eficiencia*: Los agentes pueden funcionar tanto asincrónicamente como en paralelo.
- *Robustez y confiabilidad*: La existencia de múltiples agentes presenta algunas características de tolerancia a las fallas.
- *Escalabilidad y flexibilidad*: El sistema puede ser escalado naturalmente agregando nuevos agentes.
- *Costos*: Este acercamiento descompone el problema en muchos subsistemas simples con costo unitario menor.

- *Desarrollo y reusabilidad*: Los agentes pueden ser extendidos, reutilizados, probados y mantenidos fácilmente.

Sin embargo, como todo, también presenta una serie de desventajas:

- Si se comparan los beneficios con otra tecnología, se puede eventualmente elegir la otra.
- Ausencia de un sistema de control central, lo que presenta dificultades para representar restricciones globales.
- Ausencia de una perspectiva global, ya que como los agentes toman decisiones basados en conocimientos locales, globalmente no son optimas.
- Delegación y confianza. Es necesario que las personas confíen en los agentes para así poder delegarles tareas.

2.3 Arquitectura de agentes

La arquitectura es la encargada de determinar cuales serán los mecanismos que utilizara un agente para reaccionar ante los estímulos que se le presenten [10]. Del mismo modo especifica como será el interactuar de .los diferentes módulos de un agente para lograr su objetivo. A continuación se presentan tres tipos de arquitectura:

Arquitecturas Deliberativas

Las arquitecturas deliberativas son aquellas que utilizan modelos de representación simbólica del conocimiento En este tipo de arquitectura los agentes son capaces de generar planes desde el estado inicial en que se encuentran hasta el estado final definido por su finalidad. Un agente deliberativo es aquel que tiene un modelo simbólico del mundo representado explícitamente y en donde las decisiones que tome utilizan mecanismos de razonamiento lógicos basados en la manipulación simbólica y la correspondencia de patrones con el fin de que el agente pueda alcanzar sus objetivos.

Arquitecturas Reactivas

La representación simbólica del conocimiento se caracteriza por no poseer como elemento central de razonamiento un modelo simbólico y por no incluir razonamiento simbólico complejo. A partir de esta problemática es que nacen las arquitecturas reactivas.

Una arquitectura reactiva o alternativa es aquella que no usa el razonamiento simbólico complejo y no tiene alguna clase central de modelo simbólico del mundo. Por ende las decisiones que toma una entidad están basadas en una asignación de lectura del medio con una operación a realizar.

Arquitecturas Híbridas

Cuando ninguna de las arquitecturas nombradas anteriormente cubren las necesidades para la construcción de agentes, se propone utilizar esta arquitectura que consiste en integrar la arquitectura reactiva y deliberativa. Para esto se propone crear dos subsistemas, cada uno con las características de las arquitecturas nombradas. Este tipo de arquitectura se puede desarrollar en capas donde una o más capas pueden tener acceso a los datos que entrega el entorno, mientras que otras pueden efectuar acciones al entorno.

2.3.1 Arquitectura FIPA

Se detallará esta estructura por uno de los estándares con mayor aceptación.

FIPA [8] es una colección de estándares que pretenden promover la inter-operación entre agentes heterogéneos y servicios que dichos agentes pueden representar. Gran cantidad de entornos de desarrollo orientado a agentes son compatibles con FIPA.

FIPA define la interfaz de la plataforma de agentes, las decisiones de diseño queda a criterio del equipo de desarrollo. A fin de los sistemas heterogéneos puedan interactuar nivel se sociedad de agentes, se trata de que los sistemas sean abiertos. Además establece el modelo lógico referente ala creación, destrucción, registro, localización y comunicación de agentes.

Define también los servicios que debe proporcionar toda plataforma de agentes (figura 2.1):

- Sistema de transporte de mensajes (Internal Plataform Message Transport)
- Sistema de gestión de agentes (Agent Management System)
- Servicio de directorio (Directory Facilitator)
- Canal de comunicación para agentes (Agent Comunication Channel)

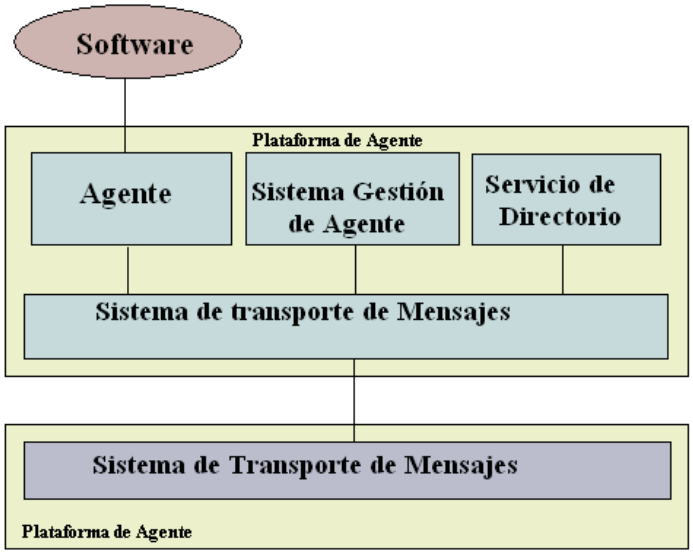

Figura 2.1: Modelo de Referencia de FIPA

2.4 Infraestructura de Agentes

La función de la infraestructura de agentes es hacer posible la interacción entre agentes. Esta infraestructura debe ser capaz de proporcionar regularizaciones para que los agentes puedan comunicarse y entenderse, de tal modo que sean capaces de compartir conocimientos. Dicha infraestructura esta compuesta por tres elementos:

- Ontologías
- Lenguajes de Comunicación
- Protocolos de Interacción

2.4.1 Ontologías

Se entiende por ontología a la comprensión compartida de algún dominio de interés. De este modo al compartir los agentes una ontología, están compartiendo conceptos y conocimientos adquiridos de modo individual lo que es altamente relevante para mantener interoperabilidad.

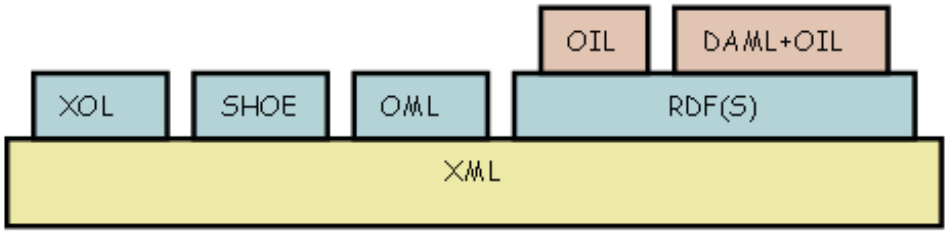

Figura 2.2: Lenguajes para Ontologías

Para poder describir una ontología existen diversos lenguajes que describen los conceptos de manera no ambigua. En la figura 2.2 se muestra como están basados estos lenguajes. Los lenguajes más comunes son:

- *XOL*: Ontology eXchange Language, US bioinformatics community.
- *SHOE*: Simple HTML Ontology Extension, University of Maryland.
- *OML*: Ontology Markup Language, University of Washington.
- *DAML*: DARPA Agent Markup Language, DARPA project (Defense Advanced Research Project Agency)
- *OIL*: Ontology Interchange Language, Onto Knowledge project (European IST)
- *DAML+OIL*: W3C.
- *OWL*: Web Ontology Language, W3C.

2.4.2 Lenguajes de Comunicación

Dentro de las características de agente, una de las más importantes es sin duda la capacidad de comunicación con otros agentes. Al tener la capacidad de comunicación, los agentes deben poder reaccionar cuando otro agente desee establecer algún tipo de comunicación. Para que esto sea posible es necesario que los agentes compartan los siguientes aspectos:

- *Sintaxis*: como se estructuran y cuales son los símbolos utilizados.
- *Semántica*: que denotan los símbolos utilizados.
- *Pragmática*: como ese interpretan los símbolos.

El significado es una combinación entre semántica y pragmática. De este modo los agentes, al comunicarse, podrán entender y ser entendidos. Esta capacidad es parte de la percepción (que recibe mensajes) y de la acción (que envía mensajes).

La comunicación humana hablada o acto del habla ha sido utilizada como modelo para la comunicación entre agentes. El acto del habla implica tres aspectos:

- *Locución:* uterancia física de quien habla.
- *Ilocución*: significado de la uterancia.
- *Perlocución*: acción que resulta de la locución.

La comunicación humana presenta ambigüedades en la locución, esto se debe a que una frase puede ser interpretada de diferentes maneras. Para evitar esto existen las "performatives" que ayudan a identificar la fuerza ilocutoria de una oración

Para la comunicación entre agentes se utilizan principalmente dos estándares: KQML y FIPA ACL.

KQML

KQML (Knowledge Query Manipulation Language) fue concebido como un formato tanto para mensajes como para un grupo de protocolos. Su finalidad es apoyar a los agentes en la identificación y conexión con otros agentes para lograr el intercambio de información.

Los principales elementos de un mensaje KQML son:

- *Performative*: Especifica el tipo de acto comunicativo del mensaje.
- *Sender:* Denota la identidad del agente que envía el mensaje.
- *Receiver*: Denota la identidad de el(los) agente(s) que reciben el mensaje.
- *Content:* Especifica el contenido del mensaje.
- *Language*: El lenguaje utilizado en el contenido.
- *Ontology:* Especifica las ontologías utilizadas.
- *Reply-with*: Identificador para ser usado por un mensaje en respuesta a este mensaje.
- *In-reply-to*: Identificador del mensaje que provocó el envío de este mensaje.

FIPA ACL

FIPA ACL es un lenguaje de comunicación de agentes que presenta claramente los elementos prácticos de la comunicación y cooperación inter-agentes, además de un fundamento semántico formal bien definido. Sus elementos principales de mensaje son muy similares a los de KQML, diferenciándose de este en el grupo de performatives predefinidos.

2.4.3 Protocolos de Interacción

"Un protocolo de interacción (AIP) describe un patrón de comunicación como una secuencia permitida de mensajes entre agentes y las restricciones al contenido de estos." [Odel, Parunak & Bauer *et al*. 2001]. Los AIP`s son los encargados de gobernar el intercambio de mensajes entre agentes, corresponden a un framework o infraestructura de alto nivel para establecer las relaciones entre agentes.

FIPA ha especificado una serie de protocolos pre-establecidos para el intercambio de mensajes, algunos de estos protocolos son:

- Suscribe: Permite a un agente (*initiator*) subscribirse a otro (*participant*), de manera que este último lo informe de acontecimientos que sean de interés del primero.
- *Request*: Se solicita a un agente realizar alguna tarea.
- *Request When*: Se solicita a un agente realizar alguna tarea al cumplirse cierta condición.
- *Contract Net*: Un agente manager solicita un presupuesto a un grupo de agentes para realizar alguna tarea, recibe dichos presupuestos y selecciona al agente que realizará la tarea.
- *Brokering:* Un broker ofrece una serie de servicios que facilitan la comunicación a otros agentes y usa su conocimiento acerca de los requerimientos y capacidades de aquellos agentes.
- *Propose*: Un agente propone una tarea a un grupo de agentes y estos pueden o no aceptar.
- *Recruiting*: Trabaja como el *brokering* pero las respuestas son directas al agente que las solicita.

2.5 Metodologías de Desarrollo Orientado a Agentes

El diseño de un sistema multiagente varía de lo tradicionalmente utilizado al insertar los conceptos de autonomía e inteligencia. Hasta hoy se han conceptualizado una serie de metodologías que pretenden modelar un sistema multiagente de manera satisfactoria, entre ellas se pueden nombrar MaSE, PASSI, Gaia, Tropos, INGENIAS, entre otras [4]. A continuación se presentaran algunas de las más utilizadas y conocidas, dando mayor énfasis a PASSI por ser el tema de este proyecto.

2.5.1 MaSE

MaSE (Multiagent Software Engineering) [7] esta basado en la orientación a objetos y asume que un agente es una especialización de un objeto, entendiendo por especialización el que los agentes se coordinan entre si mediante conversaciones y actúan proactivamente a fin de alcanzar tanto sus metas individuales como las metas del sistema. Otra definición un poco mas formal es que los agentes *"son procesos de software que interactúan entre si con el fin de alcanzar una meta global. Son solo una abstracción conveniente, que puede o no poseer inteligencia"* De este modo tanto los componentes inteligentes como los no inteligentes se tratan por igual dentro de un mismo marco.

El análisis en MaSE consiste en capturar objetivos, casos de uso y refinar roles. De cada uno de estos pasos se obtienen productos o artefactos, siendo los más importantes los siguientes:

- *Diagramas de objetivos:* Representan los requisitos funcionales del sistema.
- *Diagrama de roles*: Identifica los roles, tareas asociadas y comunicación entre ellos y las tares.
- *Casos de Uso*: Solo enumerados para utilizar mas tarde diagramas de secuencia para especificar el detalle.

Por su parte el diseño tiene cuatro pasos: crear clases de agentes, construir conversaciones, ensamblar clases de agentes y diseñar el sistema. Los artefactos de esta etapa son:

- *Diagrama de clases de agentes*: enumera los agentes del sistema, sus roles y conversaciones.
- *Diagramas de despliegue*: cuantos agentes hay y su tipo, utilizando UML
- *Descomposición de sistema*: se descompone un agente en componentes de agente y se ve la interconexión entre ellos.

2.5.2 Gaia

Gaia es una metodología para el diseño de sistemas basados en agentes cuyo objetivo es obtener un sistema que maximice alguna medida de calidad global. GAIA pretende ayudar al analista a ir sistemáticamente desde unos requisitos iniciales a un diseño que, según los autores, esté lo suficientemente detallado como para ser implementado directamente.

La metodología Gaia propone un análisis de alto nivel, utilizando solo dos modelos: el modelo de roles, que permite identificar tanto los roles claves del sistema como sus propiedades definitivas; y el modelo de interacción que define las interacciones mediante una referencia a un modelo institucionalizado de intercambio se mensajes. El diseño de Gaia define tres modelos: Modelo de agentes, que define los tipos de agentes existentes, sus instancias y su papel; Modelo de servicios que identifica las funciones de los agentes por rol; y el modelo de conocidos que define los enlaces de comunicaciones existentes entre los agentes.

A partir de aquí se propone aplicar técnicas de diseño orientadas a objetos. Gaia solo especifica como una sociedad de agentes debe colaborar para alcanzar el objetivo del sistema y detalla que se requiere de cada agente para lograr esto.

2.5.3 Tropos

TROPOS es una metodología de desarrollo de software basado en agentes que sigue tres ideas principales:

- La noción de agente y todo lo relacionado con sus nociones mentales es tomado en cuenta en todas las fases del proceso de desarrollo del sistema.
- Aborda de forma temprana la fase de análisis de requerimientos permitiendo una mayor comprensión del entorno donde operará el sistema resultante. Incluye los tipos de interacción entre usuarios y el sistema.
- Adopta un proceso de refinado de artefactos. Incluye dentro de su propuesta fases de análisis, diseño e implementación. Esto la diferencia de la mayoría de las metodologías existentes.

Las fases de TROPOS están divididas en cinco. Y denotan una concepción lógica similar a la de UP. A continuación se presentara un resumen de cada una de ellas.

- Análisis temprano de Requerimientos: Tiene por objetivo comprender el problema de la organización, se identifican las dependencias destre los usuarios y sus objetivos.
- Análisis Tardío de Requerimientos: se especifican los objetivos y sub-objetivos identificados en la etapa anterior. Así mismo se especifica lo que debe hacer el sistema dentro del ambiente operacional centrándose en las funcionalidades relevantes.
- Diseño de la arquitectura: El objetivo de esta etapa es definir la arquitectura del sistema de objetivos en base a subsistemas interconectados mediante controles de flujo.
- Diseño de Desarrollo: Orientada a la especificación de las capacidades de interrelación de los agentes. En esta etapa se suele seleccionar la plataforma de desarrollo.
- Implementación: Esta actividad sigue paso a paso todo lo definido en el Diseño de Desarrollo, estableciendo un mapeo entre el resultado y la plataforma de construcción seleccionada.

3 PASSI

3.1 Metodología PASSI

PASSI es una metodología para el diseño y desarrollo de sociedades multiagente paso-a-paso que va desde requerimientos-a-código e integra modelos y filosofías tanto desde la ingeniería de software orientada o objetos y de MAS, utilizando la notación UML como lenguaje de modelado[9]. Este último se usa al ser altamente aceptado tanto en el mundo académico como industrial. El desarrollo de la metodología PASSI cuenta con cinco modelos que se detallarán mas adelante.

En PASSI, un agente es una unidad significativa del software tanto a nivel abstracto como concreto. De acuerdo a esto, un agente es una instancia de una clase de agente[10], que es la implementación de una entidad autónoma, capaz de perseguir un objetivo mediante su autonomía, decisiones, acciones y relaciones sociales. Un agente puede tener diversos roles durante su interacción con otros agente a fin de conseguir sus metas. Un rol es una colección de tareas ejecutadas por un agente para conseguir sub-metas y describe un aspecto del ciclo de vida del agente y a menudo se relaciona a un servicio ofrecido por el agente a la sociedad o al logro de una de sus metas. Por otra parte, una tarea es definida como una entidad que apunta a alcanzar una sub-meta.

La metodología PASSI esta compuesta por cinco modelos referentes a diversos niveles de diseño: Modelo de requerimientos del Sistema, Modelo de Sociedad de Agentes, Modelo de Implementación de Agentes, Modelo de Código y modelo de despliegue. A su vez cada uno de estos modelos esta compuesto por fases, que en conjunto componen los doce pasos de la modelación de PASSI como se muestra en la figura 3.1.

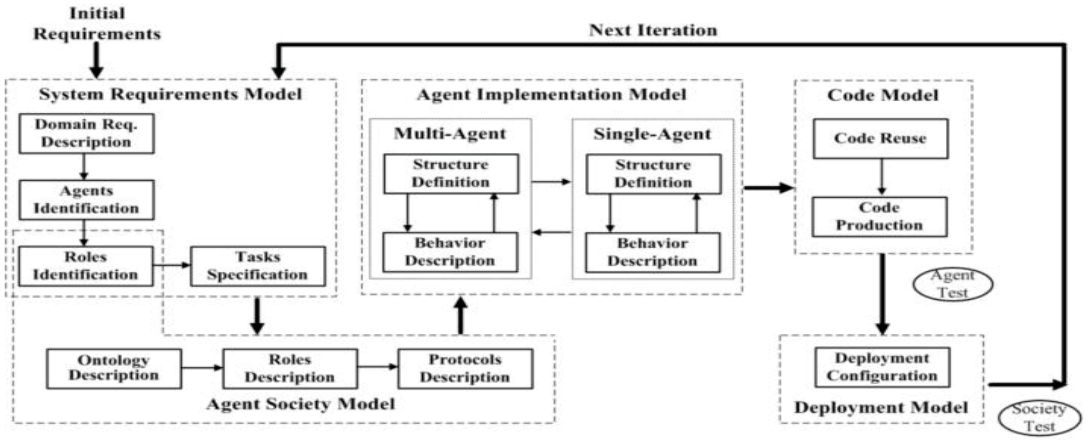

Figura 3.1: Modelo de Referencia PASSI

3.1.1 Modelo de Requerimientos del Sistema

Corresponde a un modelo antropomórfico de los requerimientos del sistema en términos de organización y propósito. Este modelo esta compuesto por cuatro fases.

3.1.1.1 Fase de Descripción de Dominio (DD)

En esta fase se produce un diagrama de casos de uso donde se identifican todas las entidades que van a interactuar con el sistema, así como todos los casos de uso que describan la funcionalidad del sistema. Siguiendo el estándar, se representaran las entidades como actores.

Los casos de uso y los actores que aparecen en esta fase, deben corresponder a los que aparecen en las distintas fases del modelado (figura 3.2). Del mismo modo, si se agrega algún caso de uso o actor durante esta fase, esta modificación debe estar reflejada en las siguientes etapas de PASSI.

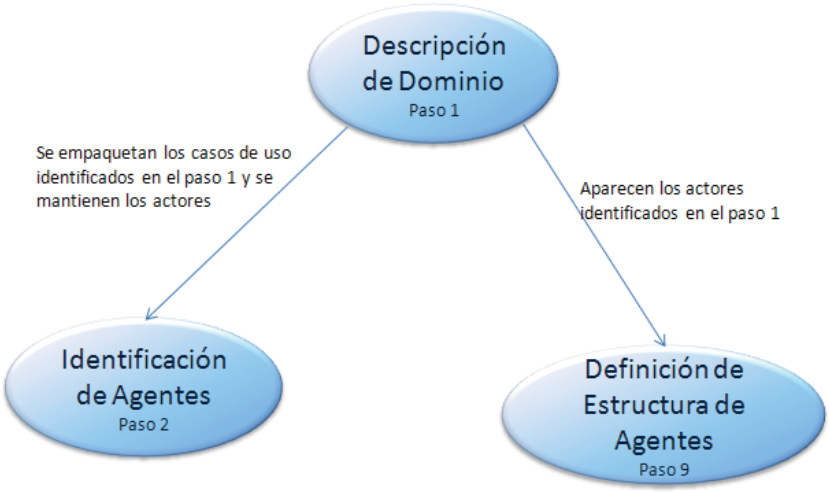

Figura 3.2: Descripción de Dominio y sus interacciones

3.1.1.2 Fase de Identificación de Agentes (I.A.)

La Identificación de Agentes corresponde a la separación de responsabilidades de los agentes y se representa agrupando casos de uso en paquetes. Una de las características de PASSI es la identificación temprana durante el proceso de desarrollo de los agentes. La identificación del agente comienza en los diagramas de caso de uso del paso anterior. De acuerdo a la definición de agentes, estos se pueden ver como casos de uso o paquetes de casos de uso (figura 3.3).

El nombre del paquete es el nombre del agente y cada paquete define una funcionalidad específica del agente. Los estereotipos de las relaciones entre los casos del uso de diversos paquetes (agentes) se convierten a ‘communicate’ puesto que diversos

agentes pueden interactuar solamente de esta manera. La dirección de las relaciones va del iniciador al participante.

Al crear un paquete se crea un agente, siendo estos agentes los mismos que aparecen en las diversas etapas de PASSI. Si en esta fase se elimina algún paquete, se eliminara también un agente, afectando los demás diagramas en caso de haber sido creados.

Figura 3.3: Identificación de agentes y sus interacciones

3.1.1.3 Fase de Identificación de Roles (I.R.)

Esta fase ocurre durante el análisis de requerimientos debido principalmente a que describe más la funcionalidad del MAS que su estructura. Se utilizan diagramas de secuencia para explorar los diferentes escenarios donde los agentes se ven interactuando y trabajando recíprocamente para alcanzar el objetivo planteado por el sistema. Los diferentes roles de un agente son introducidos como objetos dentro del diagrama de secuencia apropiado, siendo su nomenclatura ‘rol: agente’ (figura 3.4). Los agentes que participan en esta fase son los mismos identificados en la etapa anterior, del mismo modo los actores que participantes deben corresponder a los descritos en la fase de descripción de Dominio. Un mismo agente puede participar en distintos escenarios y jugando un rol diferente en cada uno de ellos, incluso puede aparecer como más de un objeto en un mismo diagrama de secuencia.

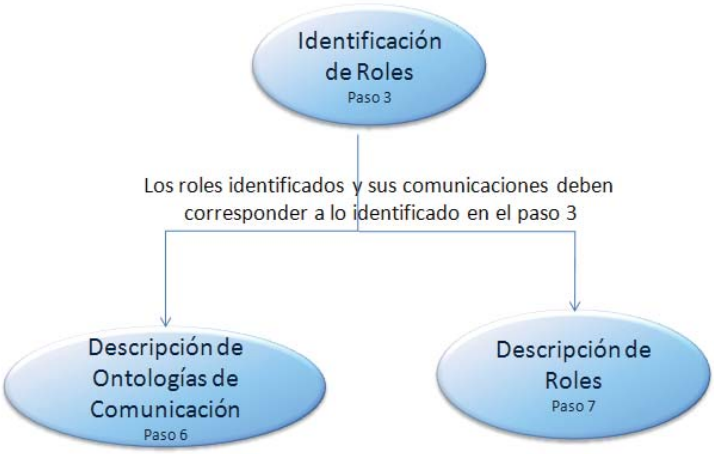

Figura 3.4: Identificación de Roles y sus interacciones

3.1.1.4 Fase de Especificación de Tareas (E.T.)

La especificación de tareas corresponde al último paso del modelo de Requerimientos de Sistema. En los pasos anteriores se han identificado los agentes y sus roles, en este fase se trabaja con un agente a la vez dibujando un diagrama de actividades por cada uno de los agente identificado previamente en la Fase de Identificación de Agentes, con el fin de representar todas las actividades que este realiza a fin de lograr su objetivo (figura 3.5). Cada diagrama esta subdividido en dos secciones: en la sección de la derecha se describen las actividades del agente al cual se esta analizando, mientras que en el de la izquierda se identifican las actividades de los demás agentes que interactúan con el agente en análisis.

Para representar las actividades en el sector izquierdo del diagrama de actividades se utiliza la siguiente nomenclatura: ‘agente.tarea’, mientras que las actividades del sector derecho del diagrama se identifican de la forma ‘tarea’.

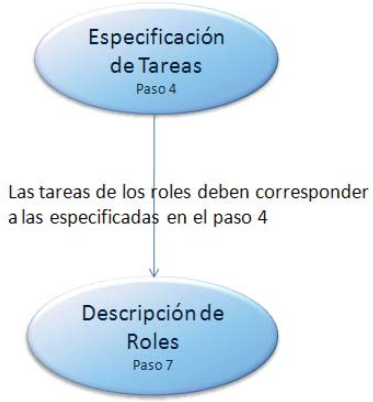

Figura 3.5: Especificación de tareas y sus interacciones

3.1.2 Modelo de Sociedad de Agentes

Es un modelo de las interacciones sociales y las dependencias entre los agentes involucrados en la solución. Se parte desde modelo anterior y esta compuesto por cuatro pasos:

3.1.2.1 Fase de Descripción de la Ontología del Dominio (D.O.D.)

En esta fase se crea un diagrama de clases que representa la ontología de dominio representando las entidades mediante clases. La ontología no solo modela los conceptos del dominio sino también las interacciones con ellos. Al hablar de interacción se hace referencia a las acciones que puede efectuar un agente con su entorno y los predicados que las describe.

La ontología se describe en términos de conceptos (entidades de dominio), predicados (valor de los atributos de predicado de dominio) y acciones (que operan en el dominio). Los elementos de la ontología pueden ser relacionados utilizando tres relaciones estándares de UML:

- Generalización: permite la relación de generalización/especialización entre dos entidades que es uno de los operadores fundamentales para construir una ontología.
- Asociación: modela la existencia de una cierta clase de relación lógica entre dos entidades.
- Agregación: puede ser utilizada para construir sets donde restricciones de valor puedan ser especificadas explícitamente.

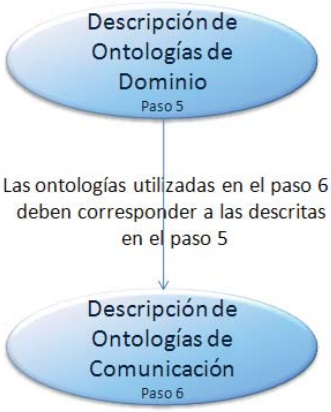

Figura 3.6: Descripción Ontologías de Dominio y sus interacciones

3.1.2.2 Fase de Descripción de la Ontología de Comunicación (D.O.C.)

El DOC corresponde a un diagrama de clases que esta compuesto principalmente de dos elementos: agentes y comunicaciones. Cada agente se describe en términos de su

conocimiento y en cada relación los roles jugados por los agentes durante la comunicación también se indican. Cada comunicación es representada por la relación entre los dos agentes y se detalla en la clase de atributos de relación. Dicha clase es identificada por un nombre y es descrita por los campos ontología, lenguaje y protocolo, donde el campo *ontology* se refiere a un elemento del tipo *concepto* del DOD (figura 3.6); el *language* se ocupa del lenguaje utilizado en el contenido de la comunicación mientras que el *protocol* precisa el protocolo de interacción (FIPA) adoptado o modificado que se detallará en la fase de Descripción de Protocolo. Los distintos roles de los agentes y la comunicación entre ellas se obtienen a partir del diagrama R.Id. En este diagrama deben aparecer cada uno de los roles identificados previamente para cada uno de los agentes, respetando las comunicaciones definidas en el R.Id.

Figura 3.7: Descripción Ontologías de Comunicación y sus interacciones

3.1.2.3 Fase de Descripción de Roles (D.R.)

La finalidad de esta fase es modelar el ciclo de vida de cada agente buscando los roles que puede jugar, las colaboraciones que necesita y las comunicaciones en las que participa. Esta información se presenta con un diagrama de clases donde los roles son clases agrupadas en paquetes que representan a los agentes. Los roles se pueden conectar por relaciones que representan cambios de rol, dependencias por un servicio o la disponibilidad de un recurso y comunicaciones. Cada rol se obtiene componiendo varias tareas, por esta razón se especifican también las tareas involucradas en el rol usando el compartimiento de las operaciones de cada clase. La dependencia entre los roles debe corresponder a una comunicación en el D.O.C., del mismo modo las tareas de los roles deben corresponder a las tareas asignadas al agente en la fase E.T.

Figura 3.8: Descripción de Roles y sus interacciones

3.1.2.4 Fase de Descripción de Protocolos (D.P.)

Aunque existen protocolos de comunicación estándares, es posible que los desarrolladores requieran especificar uno nuevo. Si este es el caso se puede describir este nuevo protocolo mediante diagramas de secuencia.

3.1.3 Modelo de Implementación de Agentes

La etapa de implementación de agentes esta formada por dos fases estrechamente relacionadas entre si: Definición de Estructura de Agente y Descripción de Comportamiento de Agente. Ambas fases además tienen dos vistas, una de agente single y una de multiagente.

3.1.3.1 Fase de Definición de Estructura de Agente (D.E.A.)

Esta compuesta por una serie de diagramas de clase. Se inicia el diseño del sistema que será implementado con ellos. Estos diagramas corresponden a dos niveles de detalle del diseño: multiagente y agentes individuales. En el primer caso se enfoca en la arquitectura general del sistema y por ende se pueden encontrar a los agentes y sus tareas. En el segundo nivel se busca la estructura interna de cada agente, mostrando todos los atributos y métodos de la clase agente así como sus clases de tareas internas. Para la arquitectura general del sistema o DEA se toma la interacción de los agentes con los actores de la fase IA, los atributos del agente de la fase DOC y los métodos de DR (figura 6.8).

3.1.3.2 Fase de Descripción del Comportamiento de Agentes

Esta fase también tiene dos niveles de abstracción. En los diagramas multiagente se representan los flujos de eventos, que invocan métodos e intercambio de mensajes entre agente, mediante diagramas de actividades. Por su parte los diagramas de agentes individuales muestran la implementación de las formalidades usadas en los métodos, se puede utilizar el método que se considera mas apropiado para describirlo.

3.1.4 Modelo de Código

Corresponde al código de la solución. Es aquí donde se emplea la reutilización de patrones Fase de Librería de Reutilización de Código. En esta fase se trata de reutilizar patrones de agentes y tareas ya existentes. Estos patrones no son solo trazos de código, también son trozos de diseño que pueden ser usados en distintos sistemas. A nivel de programación, el diseñador esta profundamente involucrado en resolver los detalles de la implementación de varios agentes y no tiene una visión completa del sistema. El programador debe completar el código de la aplicación comenzando desde el diseño, al esqueleto generado y los patrones utilizados.

3.1.5 Modelo de Despliegue

3.1.5.1 Configuración de Despliegue

Se utiliza un diagrama del despliegue para describir la diseminación de los agentes a través de las plataformas disponibles y de sus movimientos. Las plataformas se describen como nodos de proceso, los agentes como componentes y sus comunicaciones (si no son demasiados) conectan al iniciador con el símbolo de interfaz del participante. Representamos los movimientos del agente con una relación desde el agente en la posición anterior al agente en la posición de destino. A menudo se agregan notas a estas relaciones para especificar las pre-condiciones para el movimiento del agente.

3.2 PASSI ToolKit

La herramienta PASSI ToolKit o PTK [6] es un plug-in de Rational Rose que ofrece soporte a la metodología PASSI mediante el diseño y desarrollo de los sistemas multiagente paso a paso, permitiendo al desarrollador generar los diagramas que pide cada modelo de las etapas de PASSI.

Es una aplicación de difusión comercial basada en UML que entrega un diseño robusto y coherente disminuyendo el esfuerzo del diseñador. Además entrega soporte a la fase de producción de código al permitir auto generar una gran cantidad de código. PTK esta compuesta de dos herramientas que interactúan entre si. La primera, llamada *PASSI Add-in,* es una aplicación que funciona como una extensión de Rational Rose y que le permite a esta última soportar la metodología PASSI, permitiendo al usuario implementar

las diferentes fase de PASSI gracias a un compilado automático de los diagramas. Esta parte del PTK también entrega una interfaz para administrar el repositorio de patrones.

La segunda herramienta de PTK, llamada *Repositorio de Patrones*, entrega un repositorio de patrones junto con un buscador para patrones recogidos e integrados con el sistema multiagente que se esta desarrollando.

Para el propósito de este informe se analizará a continuación únicamente el *PASSI Add-in*, por ser el tipo de herramienta que se esta desarrollando durante este proyecto.

3.2.1 Inconvenientes con PASSI Add-In

Este add-in ofrece un grupo de funcionalidades que hace posible la automatización de algunas de las operaciones requeridas en el proceso desarrollo de softwares multiagente, por ejemplo: la compilación total o parcial de diagramas, la identificación de agentes y sus tareas, entre otras. Esta herramienta posee una interfaz grafica adecuada que permite la comprensión de las tareas a realizar, sin embargo no es lo suficiente clara para explicar la función de cada una de sus tareas.

Para comenzar a usar el PASSI Add-in es necesario agregarlo al listado de plug-in y activarlo. Una vez hecho esto se puede comenzar a crear los diagramas. Es altamente sugerido hacer un desarrollo lineal del MAS con esta herramienta, ya que tiene altas posibilidades de crear inconsistencias entre los diagramas si se realizan modificaciones en etapas que ya han sido pasadas y chequeadas. Durante todo el desarrollo del MAS, al pasar de un diagrama a otro se realiza un chequeo de consistencia del diagrama que se ha finalizado, sin embargo, el listado de los errores se entrega en un log file que puede ser fácilmente, pudiendo así pasar al siguiente diagrama arrastrando errores que causarán inconsistencias en el MAS.

Un ejemplo que se puede dar de los detalles que presenta PTK es en la fase de Descripción de Dominio. Al finalizar la identificación de los actores y todos los casos de uso involucrados en el sistema se deben seleccionar uno a uno cada caso de uso y comunicación perteneciente a un agente, siendo esta selección engorrosa y que fácilmente provoca el tener que realizar nuevamente la selección al quedar algún elemento fuera por falta de claridad en el dibujo.

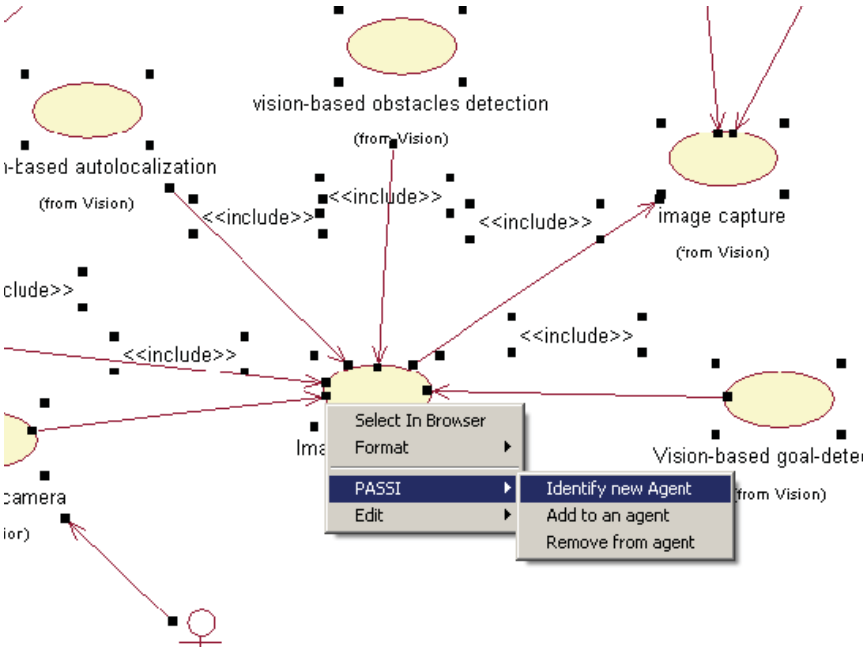

Figura 3.9: Creación de agentes

Otro ejemplo claro es la incapacidad que tiene el desarrollador para organizar la ubicación física en el diagrama de Identificación de Agentes, de los casos de uso y los agentes que aparecen él. Como este diagrama es autogenerado por PTK, la ubicación de cada elemento que hay en el no se puede alterar. Si el desarrollador 'mueve algún elemente', este cambio se perderá al cerrar el PTK.

Un inconveniente importante es la poca estabilidad de la herramienta, esto se debe a la gran cantidad de recursos que utiliza Rational, lo que deriva en que, a menos que se cuente con un buen procesador, el realizar otra tarea mientras se trabaja con PTK puede generar la 'caída' del programa provocando perdidas de información.

A pesar de que el PTK se encuentra disponible para ser descargado de forma gratuita en diferentes sitios de Internet, solo puede ser empleado en una herramienta CASE de distribución comercial, lo que implica tener que comprar dicha herramienta para utilizar el add-in

A pesar de que PTK cuenta con una pagina Web donde esta disponible el manual de usuario y la aplicación para ser descargada de forma gratuita, no cuenta con algún tipo de foro o información extra para ayudar a los diseñadores a trabajar de mejor forma o a solucionar las diferentes problemáticas que se generen durante el proceso de modelado. Así mismo, en Internet, es bastante difícil encontrar información mas acabada sobre PTK por ser una herramienta nueva y para una metodología no tan conocida.

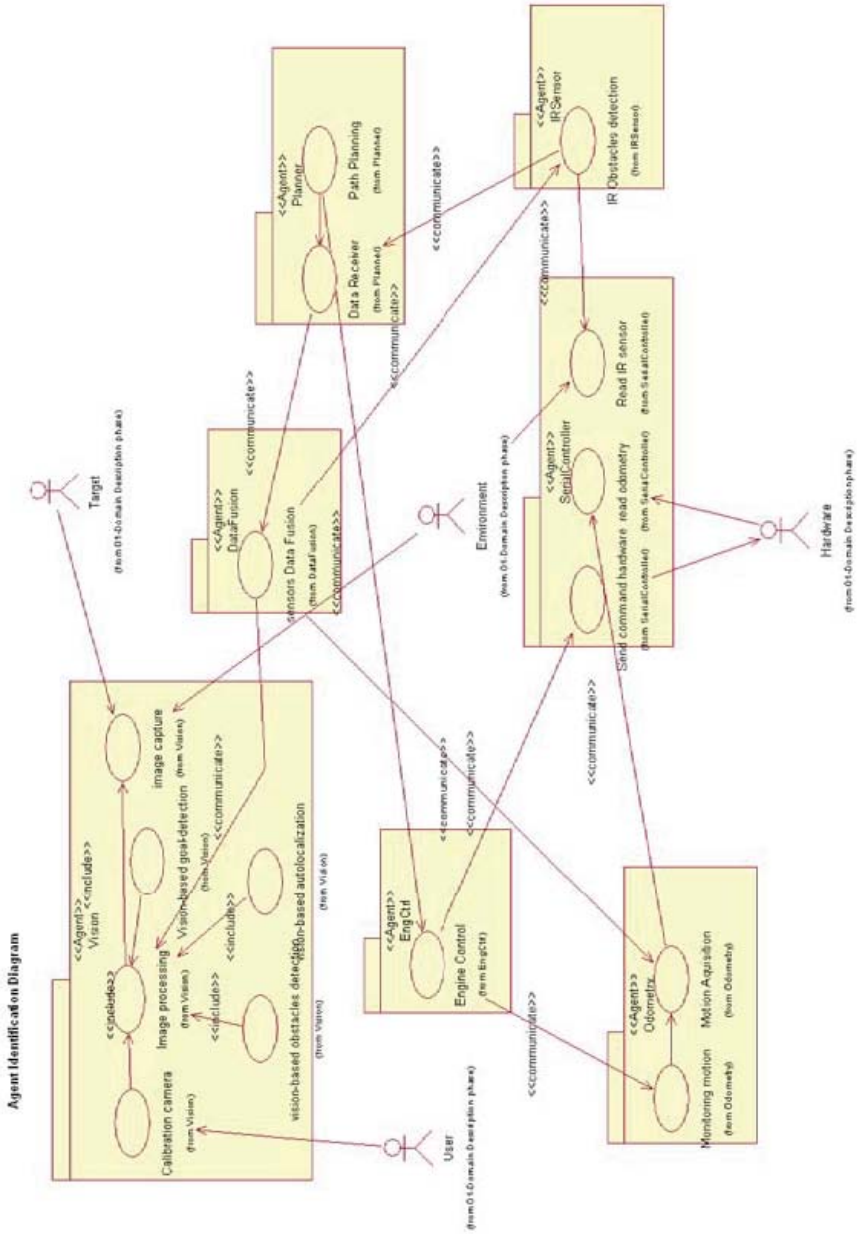

Figura 3.10: Autogeneración de diagrama de Identificación de Agentes

4 Eclipse

4.1 IDE

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que se llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores de Internet. Esta plataforma se ha usado para desarrollar entornos de desarrollo integrados (IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son utilizan a su vez para desarrollar el mismo Eclipse). Pero no solo eso, también es una comunidad de usuarios, lo que permite extender constantemente las áreas de aplicación cubiertas.

Fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge, sin embargo ahora es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Plataforma

La plataforma de Eclipse esta diseñada y construida par cumplir con lo siguiente:

- Soporte a la construcción de diversas herramientas para el desarrollo de aplicaciones
- Soporte a un grupo de proveedores de herramientas no restringida incluyendo ISV´s (Independent Software Vendors).
- Soporte de herramientas para la manipulación arbitraria de tipos de contenido (HTML, Java, C, JSP, EJB, XML, GIF).
- Facilita la integración de herramientas que poseen o contiene distintos tipos de contenidos y proveedores de herramientas.
- Soporta entornos de desarrollo tanto basadas o no basadas en GUI.
- Funciona bajo un amplio rango de sistemas operativos, incluyendo Windows y Linux.

El principal rol de la plataforma Eclipse es entregar tool providers con mecanismos para usar, reglas para seguir y que lleven a la integración de herramientas. Estos mecanismos están expuestos vía interfaces API bien definidas, clases y métodos. La plataforma también entrega útiles frameworks que facilitan el desarrollo de nuevas herramientas.

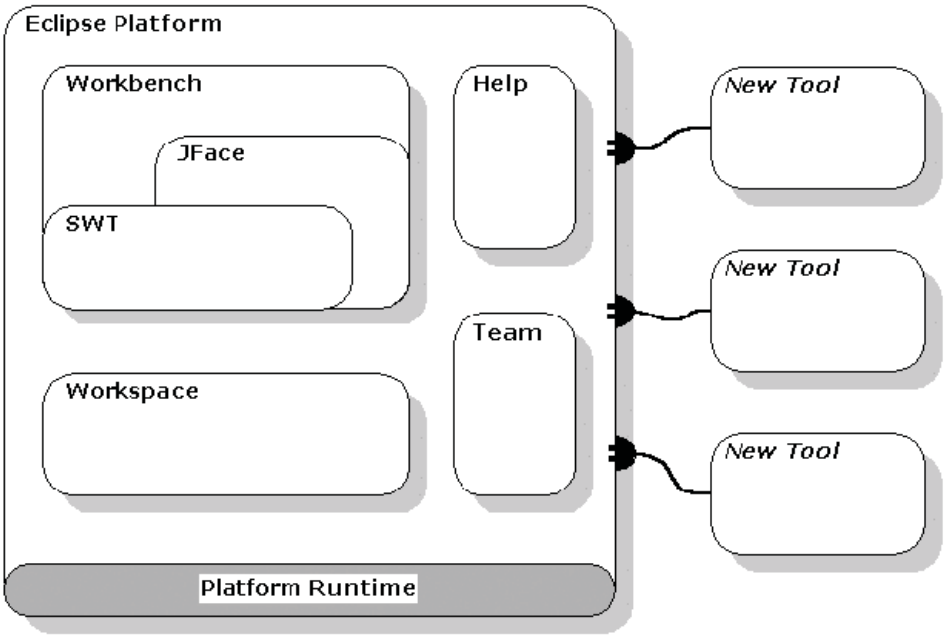

Figura 4.1: Arquitectura de la Plataforma Eclipse

Plug-in

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente al permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse.

Frameworks

Eclipse provee al programador con frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, GEF (Graphic Editing Framework - Framework para la edición gráfica) es un plugin de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto wysiwyg hasta editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF "viven" dentro de Eclipse, además de poder ser usados conjuntamente con otros plugins, hacen uso de su interfaz gráfica personalizable y profesional.

SDK

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

4.2 Violet

Violet es un plug-in en Java desarrollado en Eclipse 3.1.1 en adelante que puede ser ejecutado en cualquier plataforma que tenga JRE 6.0 o superior y tiene la capacidad de utilizarse como una aplicación stand-alone, como un applet, vía Java Web Start o como un plug-in de Eclipse.

Es un editor de UML que cuenta con los siguientes beneficios: es muy sencillo de aprender y usar, dibuja diagramas agradables a la vista, es completamente gratis y es multiplataforma. Violet está enfocado en desarrolladores, estudiantes, profesores y autores que necesitan crear rápidos y simples diagramas UML.

Cuando Violet es utilizado como plug-in de Eclipse, tiene la ventaja de utilizar todas las características que proporciona Eclipse: soporta y permite la utilización del historial local de Eclipse y restaurar diagramas realizados anteriormente, permite convertir clases Java en diagramas de secuencia y diagramas de clase, y permite drag&drop diagramas existentes al diagrama actual para crear vínculos entre diagramas.

Características

Violet permite al usuario generar los siguientes tipos de diagramas: Diagramas de casos de uso, Diagramas de clase, Diagramas de actividades, Diagramas de secuencias, Diagramas de estado y Diagramas de objeto.

Sin embargo, hay algunas características que muchos de los programas orientados a modelar UML poseen y Violet no tiene.

- Generación de código. Violet no permite la generación de ningún tipo de código a partir de los diagramas UML.
- Ingeniería inversa. No es posible generar diagramas UML a partir de código fuente (excepto con los diagramas de clase de Eclipse)
- Verificación de la semántica de los modelos. Se pueden crear modelos contradictorios o con grandes errores ya que no se puede verificar la consistencia de dichos diagramas.
- Exportar o Importar a XML. No se pueden generar archivos para ser importados a otras herramientas UML así como tampoco se pueden leer archivos con modelos desde otras herramientas.

Una característica muy importante que no posee Violet y que se requiere para el desarrollo de este proyecto es la interacción entre diagramas. Cada uno de los diagramas que genera Violet es independiente, esto es, que no existe ningún tipo de inter-relación entre los diagramas, esto mismo es lo que no permite obtener información de diagramas existentes para generar nuevos diagramas.

Sin embargo, al ser Violet un programa de código abierto que se encuentra disponible para desarrollar nuevas características sobre su base, realizar modificaciones y reutilizar su código, es factible realizar las modificaciones necesarias para lograr el objetivo de este proyecto.

Considerado todo lo mencionado anteriormente es que se decidió utilizar Violet como punto de partida para la creación de la herramienta de modelado de la metodología PASSI.

5 Desarrollo del Proyecto

En este capítulo se mostrará el trabajo desarrollado hasta la fecha de entrega del presente informe. Como se mencionó anteriormente se utilizó el paradigma de proceso unificado. A fin de lograr una mejor comprensión de las tareas desarrolladas se separará el trabajo de acuerdo a los flujos de trabajo de UP sin subdividir en fases o iteraciones. A la entrega de este informe el proyecto se encuentra en la fase de construcción en su 3º iteración.

5.1 Requerimientos

El total de los requerimientos fueron identificados durante la fase de inicio y elaboración. Estos estan agrupados según el mode lo FURPS+ de proceso unificado siendo estos los siguientes:

Requerimientos funcionales

- Crear, modificar y eliminar agentes: el sistema debe permitir al usuario crear, modificar y eliminar agentes únicamente en la fase de identificación de agentes, no permitiendo eliminar un agente que contenga casos de uso.
- Crear, modificar y eliminar casos de uso: el sistema debe ser capaz de crear casos de uso consistentes con todas sus características, permitiendo a su vez que estos sean modificables o eliminados durante la fase de Descripción de Dominio manteniendo la consistencia de los diagramas.
- Crear y modificar diagramas de descripción de dominio: estos deben poder crearse utilizando diagramas de caso de uso de forma completa, correcta y consistente. Del mismo modo las modificaciones efectuadas en este diagrama deben reflejarse en todo el modelado sin crear inconsistencias.
- Crear y modificar diagramas de identificación de agentes: el usuario debe poder agrupar casos de uso en paquetes a fin de crear los agentes. Tanto los casos de uso como los agentes creados deben poder moverse de posición dentro del diagrama y mantener dicha posición.
- Crear y modificar diagramas de Identificación de Roles: el usuario debe poder crear y modificar diagramas de secuencia asociados a la identificación de Roles pudiendo utilizar como información los agentes creados en la fase IA pudiendo agregar tareas de cada rol creado.
- Crear y modificar diagramas de Especificación de Tareas: el usuario debe poder crear y modificar los diagramas de actividades correspondientes a esta etapa. Se debe exigir que se cree un diagrama de actividades por cada agente.

- Crear y modificar descripción de ontologías de dominio: el usuario debe poder crear y modificar diagramas de clases permitiendo agregar los conceptos predicados y acciones involucrados en ontología.

- Crear y modificar diagramas de descripción de ontología de comunicación: el usuario debe poder crear y modificar diagramas de clase asociados a esta fase. Tanto los roles de agentes como las comunicaciones entre ellos deben corresponder a las identificadas en la fase IR. Del mismo modo, las ontologías (conceptos) son tomados de la fase DOD.
- Crear y modificar diagramas de Descripción de Errores: permitir al usuario crear y modificar los diagramas de clase utilizados en esta etapa. Los roles de los agentes y las comunicaciones entre ellos deben corresponder a las descritas en la fase IR mientras que las comunicaciones entre agentes debe corresponder a las descritas en la fase DOC.
- Crear y modificar diagramas de Descripción de Protocolos: en caso de ser necesario se podrán crear y modificar diagramas de secuencia que determine la gramática de cada protocolo de comunicación. La correspondencia será un diagrama por cada protocolo.
- Crear y modificar diagramas de Definición de Estructura de Agente (multi): permitir al usuario crear y modificar un diagrama de clases para describir la estructura del sistema, donde los actores deben corresponder a los identificados en la fase DD, los agentes a la fase IA, las tareas a las descritas en la fase ET, los métodos de los agentes a la fase DR y los atributos de los agentes a la fase DOC.
- Crear y modificar diagramas de Definición de Estructura de Agente (single): permitir al usuario crear y modificar un diagrama de clases por cada agente identificado en la fase IA para describir la estructura interna del agente donde los agentes, las tareas y los conceptos serán tomados tanto de DR como de DEA (multi).
- Crear y modificar diagrama de comportamiento de agentes: permitir al usuario crear y modificar los diagramas de actividades asociados a esta fase tanto para agentes individuales como multiagente donde las tareas serán tomadas de DR y de DEA (single y multi).
- Actualizar diagramas: el usuario debe poder actualizar los diferentes diagramas a fin de que se apliquen los cambios realizados en forma consistente.
- Exportación de imágenes: el usuario deberá poder exportar los diagramas generados en formato de imagen.
- Chequear consistencia: el usuario debe poder chequear la consistencia del sistema que se esté creando.

Requerimientos de Usabilidad

El sistema contará con una guía de ayuda al usuario corta y precisa que explique el funcionamiento principal y las características mas importantes del sistema desarrollado. Esta guía seguirá el patrón utilizado y descrito en Eclipse.org

Requerimientos de Fiabilidad y Rendimiento

En caso de caída o fallo del sistema durante su ejecución este debe ser capaz de guardar los últimos cambios guardados a fin de perder la menor cantidad posible de modificaciones. Se prevé que el porcentaje de error del sistema debe ser inferior a 0,05%. Del mismo modo el sistema permitirá la generación y modificación de diagramas en un tiempo breve, así como libre de errores y consistente entre sí.

Requerimientos de Soporte

El código del sistema a generar debe estar especificado y documentado a fin de facilitar los cambios o modificaciones de éste; además, los modelos a generar deben estar apegados al estándar UML. El código debe contar con comentarios descriptivos a fin de facilitar la modificación del código mismo.

Requerimientos de Implementación

El sistema se creará como un plug-in en Java bajo el ID Eclipse Ganymede 3.4.0 versión RCP y se trabajará con el plug-in Violet para Eclipse.

Requerimientos de Interfaces

Las interfaces del sistema corresponderán a las estereotipadas por Violet. Deben ser sencillas e intuitivas para el usuario.

Requerimientos legales

Al ser el ID Eclipse gratuito y de código abierto, no se requiere licencia alguna ni para el ID en sí ni para los plug-in que se utilizarán.

5.2 Análisis

En esta etapa, se analizó la forma de enfrentar la problemática y la forma de enfrentar las posibles dificultades asociadas a un desarrollo de proyecto. El análisis de factibilidad económica y gestión de riesgo se encuentran descritas en el punto 4 del presente informe.

En cuanto a los plug-in que se utilizarán en el diseño del sistema se dará a continuación una pequeña descripción de ellos.

Draw2D

El plug-in Draw2D entrega a eclipse un stand-alone que permite renderizar (genera una imagen a partir de un modelo) y un paquete de administra las disposiciones físicas de los objetos. A pesar de que Draw2D puede ser usado como un stand-alone, no es un marco de trabajo para editar, la mayoría de las aplicaciones utilizan el plug-in GEF. El Draw2D incluye las formas más comunes así como tipos de líneas que se pueden combinar para renderizar casi cualquier cosa.

Algunas de las características de Draw2D son las siguientes:

- Permite implementación de varias figuras y en diversas disposiciones.
- Soporta figuras con bordes
- Permite mediante eventos simples moverse entre figuras
- Sistema de coordenadas flexible
- Opción de vista previa
- Opción de impresión
- Entre otras.

GEF

El Graphical Editing Framework permite a los desarrolladores crear editores ricos gráficamente a partir de un modelo de aplicación ya existente. Utiliza el Draw2D que le entrega las herramientas de renderizacion y disposiciones o ubicaciones, a partir de esto el desarrollador es capaz de de tomar las operaciones mas comunes de que entrega GEF y extenderlas aun dominio especifico. GEF utiliza una estructura MVC (modelo-vista-controlador) que permite que los pequeños cambios sean aplicados al modelo desde la vista. A continuación se entregan algunas de sus características más relevantes.

- Herramientas como Selección, Creación, Conexión
- Una paleta para desplegar las herramientas nombradas
- Modificar el tamaño de los objetos y reubicar las conexiones
- Dos tipos de vista: como grafico y como árbol
- Comandos de hacer y deshacer
- Controlador de framework para mapear desde al modelo de negocios a una vista.

Violet

El plug-in Violet, como se explicó anteriormente, será usado como un plug-in de Eclipse. Violet utiliza tanto GEF como Draw2D para la generación de diagramas.

5.3 Diseño

Durante esta etapa se toman las decisiones de diseño y se especifican los casos de uso. Asimismo se detalla el plan de trabajo para las siguientes iteraciones y se decide sobre el alcance del proyecto de acuerdo a la experiencia obtenida durante este proceso. El detalle de los alcances del proyecto se encuentran definidas en el punto 2.4 del presente informe, asimismo el plan de trabajo se encuentra definido en el Capítulo 3 del mismo.

Selección de IDE

Para el desarrollo del sistema se analizaron dos programas de software que presentaban características adecuadas para implementa el plug-in a desarrollar. Estos softwares son NetBeans y Eclipse. Ambos software son libres, de código abierto, y tienen entre sus add-in uno dedicado a la creación de plug-ins para su propio entorno, están generados en Java y poseen gran estabilidad así como una gran variedad de documentación de referencia y ayuda.

Sin embargo se optó por utilizar Eclipse debido, principalmente por la familiaridad ya existente con dicho IDE por parte de los desarrolladores, además de que utiliza menor cantidad de recursos de sistema y sus interfaces son sencillas e intuitivas.

Herramientas a Utilizar

El lenguaje seleccionado para crear el plug-in es JAVA y se utilizará el software Eclipse por ser el mismo para el cual se desarrollara dicho plug-in, además, Eclipse es un software libre y conocido por los desarrolladores. Así mismo se hará uso del plug-in Violet para Eclipse.

5.3.1 Casos de Uso

Se entrega a continuación el diagrama general de casos de uso (figura 5.1) y un ejemplo de los casos de uso mas relevante.

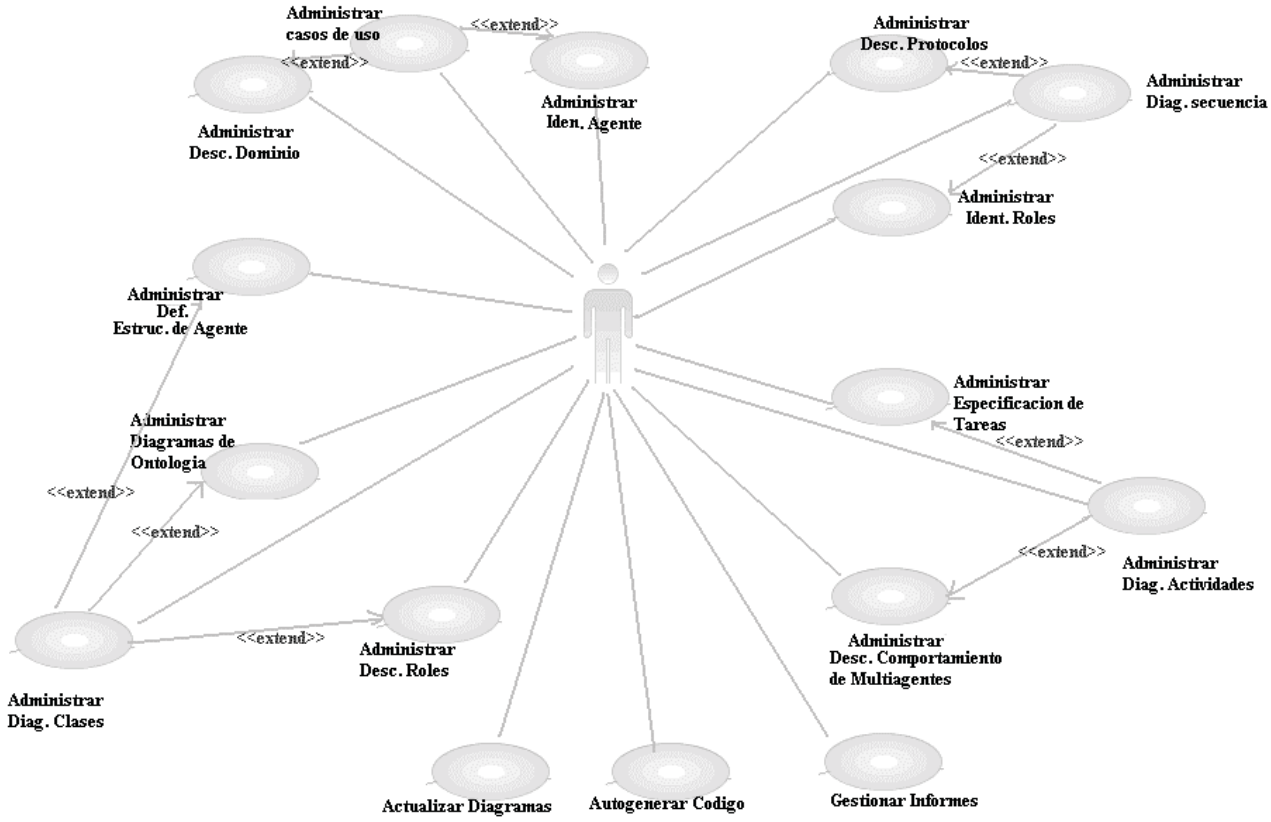

Figura 5.1: Modelo General de casos de uso

Administrar Caso de Uso (figura 5.2)

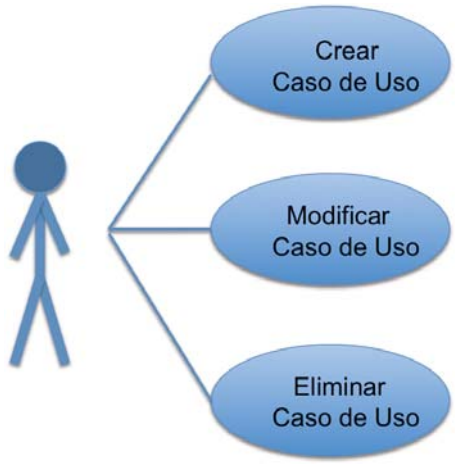

Figura 5.2: UseCase 'Administrar Caso de Uso'

Caso de Uso	***Crear Caso de Uso***
Pre-condiciones	----------
Actor Principal	*Usuario*
Escenario Principal	*1- Usuario selecciona en la paleta el icono "caso de uso" y lo ubica en la plantilla* *2- Sistema solicita un nombre para el objeto, ofrece nombre genérico.* *3- Usuario introduce nombre y acepta* *4- Sistema genera un objeto del tipo "caso de uso" en la ubicación seleccionada por el usuario* *5- Fin*
Post-condiciones	*Se ha generado y ubicado en la plantilla un objeto del tipo "caso de uso"*
Escenario Alternativo	*suario selecciona en la paleta el icono "Use case" y lo ubica incorrectamente en la plantilla.* *istema anula la acción* *uelve al paso 1*
	3- Usuario no introduce nombre y acepta *4- Sistema genera un objeto del tipo "caso de uso" en la ubicación seleccionada por el usuario con el nombre genérico ofrecido en el paso 2*
	3- Usuario cancela la operación *4- Sistema anula la acción realizada*

Administrar Desc. Dominio (figura 5.3)

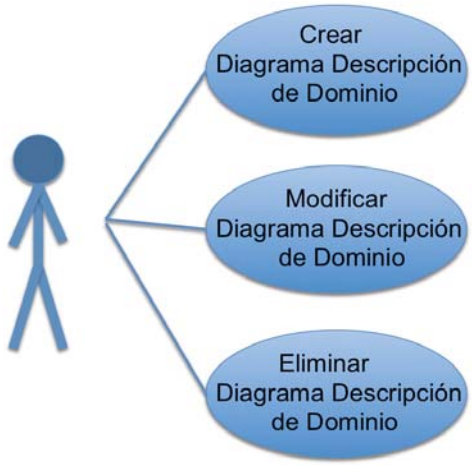

Figura 5.3: UseCase 'Administrar Diagrama Descripción de Dominio

Caso de Uso	***Generar Diagrama Descripción de Dominio***
Pre-condiciones	*Se requiere haber creado al menos un objeto del tipo "caso de uso"*
Actor Principal	*Usuario*
Escenario Principal	*1- Usuario selecciona en la paleta el icono "Actor" y lo ubica en la plantilla* *2- Sistema solicita un nombre para el objeto, ofrece nombre genérico.* *3- Usuario introduce nombre y acepta* *4- Sistema genera un objeto del tipo "actor" en la ubicación seleccionada por el usuario* *5- Usuario selecciona el icono "Interaction" y relaciona un objeto "actor" con uno del tipo "caso de uso"* *6- Sistema genera la relación* *7- Vuelve al paso 1*
Post-condiciones	*Se ha generado un Diagrama de Identificación de Agentes*
Escenario Alternativo	*suario selecciona en la paleta el icono "Actor" y le da una ubicación incorrecta en la plantilla.* *istema anula la acción* *uelve al paso 1*
	5- Usuario no introduce nombre y acepta *6- Sistema genera un objeto del tipo "actor" en la ubicación seleccionada por el usuario con el nombre genérico ofrecido en el paso 2* *8- Usuario selecciona el icono "Interaction" y relaciona un objeto "actor" con uno del tipo "caso de uso"* *9- Sistema genera la relación*

	5- Usuario no introduce nombre y acepta *6- Sistema genera un objeto del tipo "actor" en la ubicación seleccionada por el usuario con el nombre genérico ofrecido en el paso 2* *7- Usuario selecciona el icono "Interaction" y relaciona incorrectamente dos objetos* *8- Sistema anula la relación* *9- Vuelve al paso 7*
	5- Usuario cancela la operación *6- Sistema anula la acción realizada*

Administrar Iden. Agente (figura 5.4)

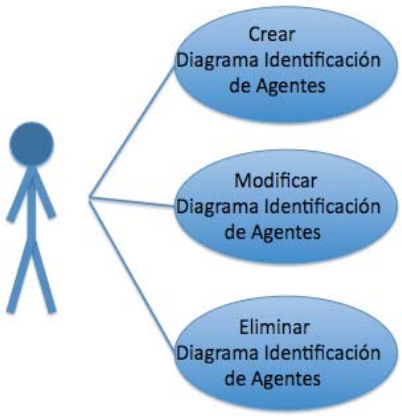

Figura 5.4: UseCase 'Administrar Diagrama Identificación de Agentes

Caso de Uso	***Generar Diagrama Identificación de Agentes***
Pre-condiciones	*Se requiere haber creado al menos un diagrama de Descripción de Dominio*
Actor Principal	*Usuario*
Escenario Principal	*1- Usuario selecciona en la paleta el icono "Agrupar"* *2- Sistema solicita seleccionar los casos de uso que serán agrupados.* *3- Usuario selecciona los casos de uso a agrupar y acepta* *4- Sistema genera un paquete con los caos de uso seleccionados* *5- Vuelve al paso 1*
Post-condiciones	*Se ha generado un Diagrama de Identificación de Agentes*
Escenario Alternativo	*3- Usuario no selecciona casos de uso* *4- Sistema anula la acción* *5- Vuelve al paso 1*

Administrar Iden. Roles (figura 5.5)

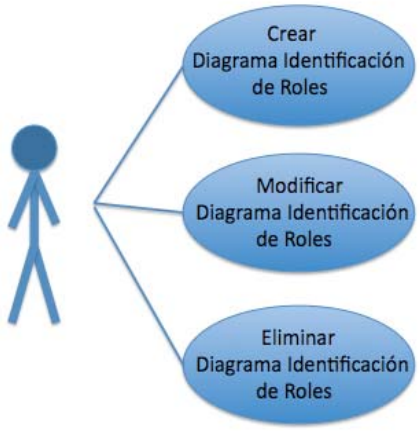

Figura 5.5: UseCase 'Administrar Diagrama Identificación de Roles

Caso de Uso	***Generar Diagrama Identificación de Roles***
Pre-condiciones	*Se requiere haber creado al menos un diagrama de Identificación de Agentes*
Actor Principal	*Usuario*
Escenario Principal	*1- Usuario selecciona el agente requerido* *2- Sistema despliega solicita seleccionar los casos de uso que serán agrupados.* *3- Usuario selecciona los casos de uso a agrupar y acepta* *4- Sistema genera un paquete con los caos de uso seleccionados* *5- Vuelve al paso 1*
Post-condiciones	*Se ha generado un Diagrama de Identificación de Roles*
Escenario Alternativo	*3- Usuario no selecciona casos de uso* *4- Sistema anula la acción* *5- Vuelve al paso 1*

5.3.2 Diagrama de Secuencia

Se generaron diagramas de secuencia como parte del desarrollo Proceso Unificado para asi ver la funcionalidad desde el punto de vista dinamico del sistema.

En la figura 5.6 se muestra uno de estos diagramas que muestra la generacion de objetos dentro de la aplicación. Esta generacion de objetos es utilizada en cada uno de los diagramas que se crean.

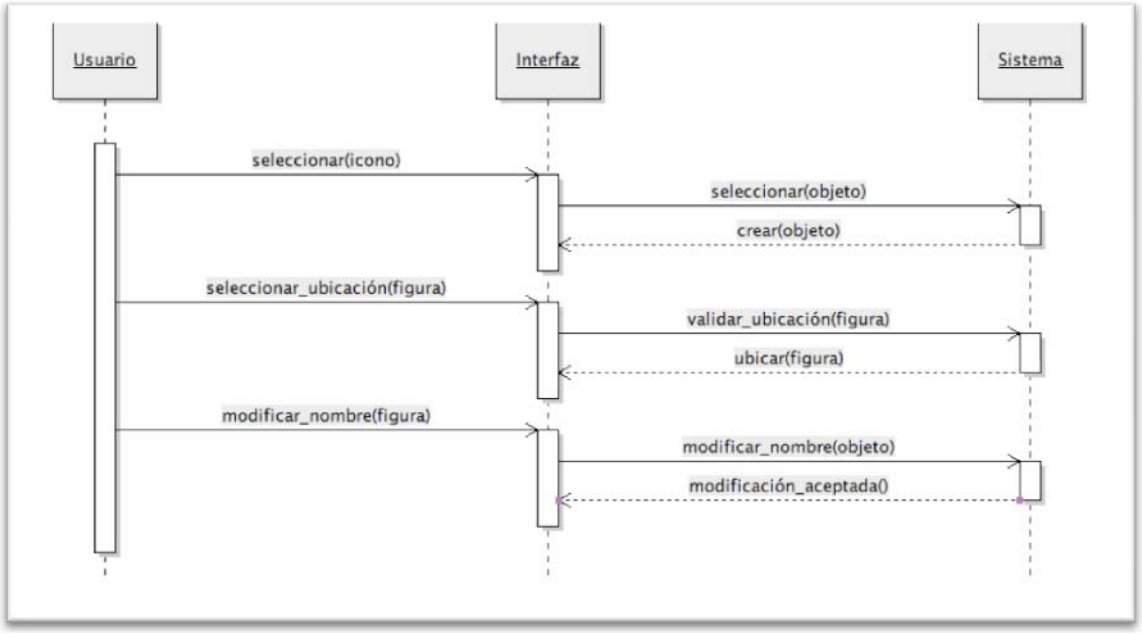

Figura 5.6 : Diagrama de Secuencia

Validador de Diagramas

Con el fin de asegurar la integridad de los diagramas se creó una función validadora que tiene por objetivo validar cada parte de los diagrama generados.
Esta funcion es llamada por el usuario cada vez que presiona el icono “validar” en la aplicación. Esta validacion se hace tanto en el diagrama en el cual se esta trabajando actualmente, como en los diagramas creados previamente asegurando asi que las modificaciones que se hayan realizado sean consideradas y los diagramas mantengan su integridad. En la figura 5.7 se muestra el diagrama de clases de esta función.

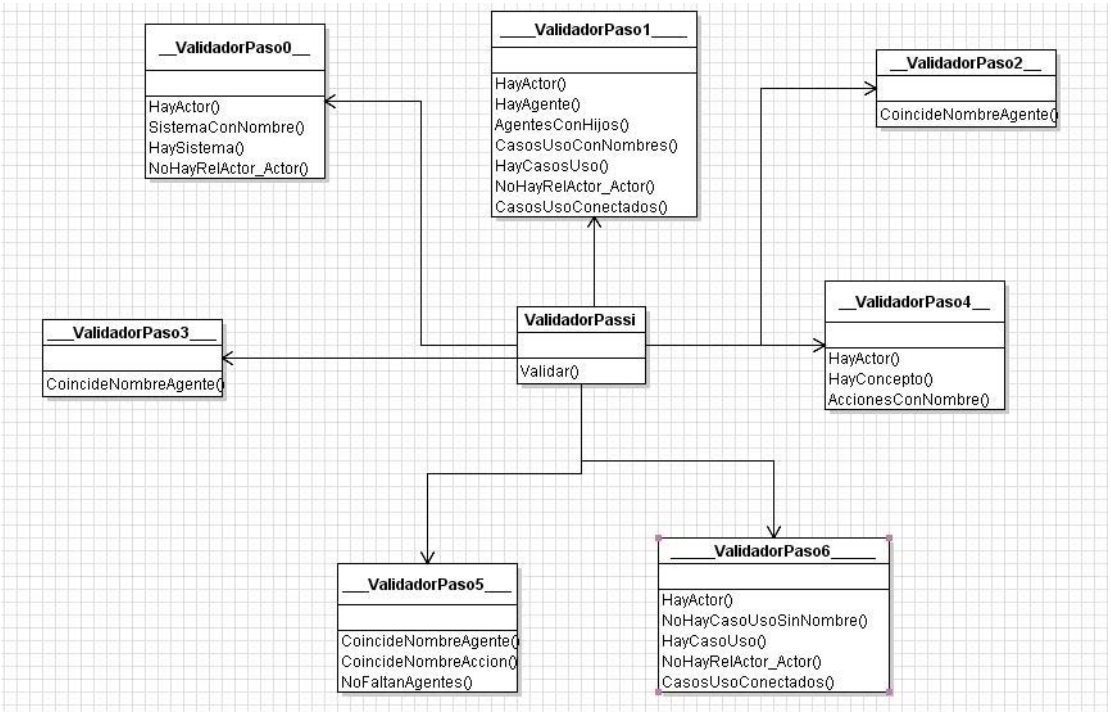

Figura 5.7: Diagrama de Clases Validador

5.4 Implementación

En la etapa de implementación se logró la creación, durante las diversas fases de UP, un prototipo funcional del sistema que es capaz de realizar casi en un 100% el modelo de requerimientos del sistema que comprende las fases de descripción de dominio, identificación de agentes, identificación de roles y especificación de tareas. El prototipo es capaz de generar los diagramas requeridos manteniendo la integridad de éstos.

5.4.1 Generacion del archivo .jar

Para generar el archivo .jar que permitira correr la aplicación hay que seguir los siguientes pasos:

1) Con el Proyecto abierto, ir al menú "File" y seleccionar la opción "Export" (figura 5.8)

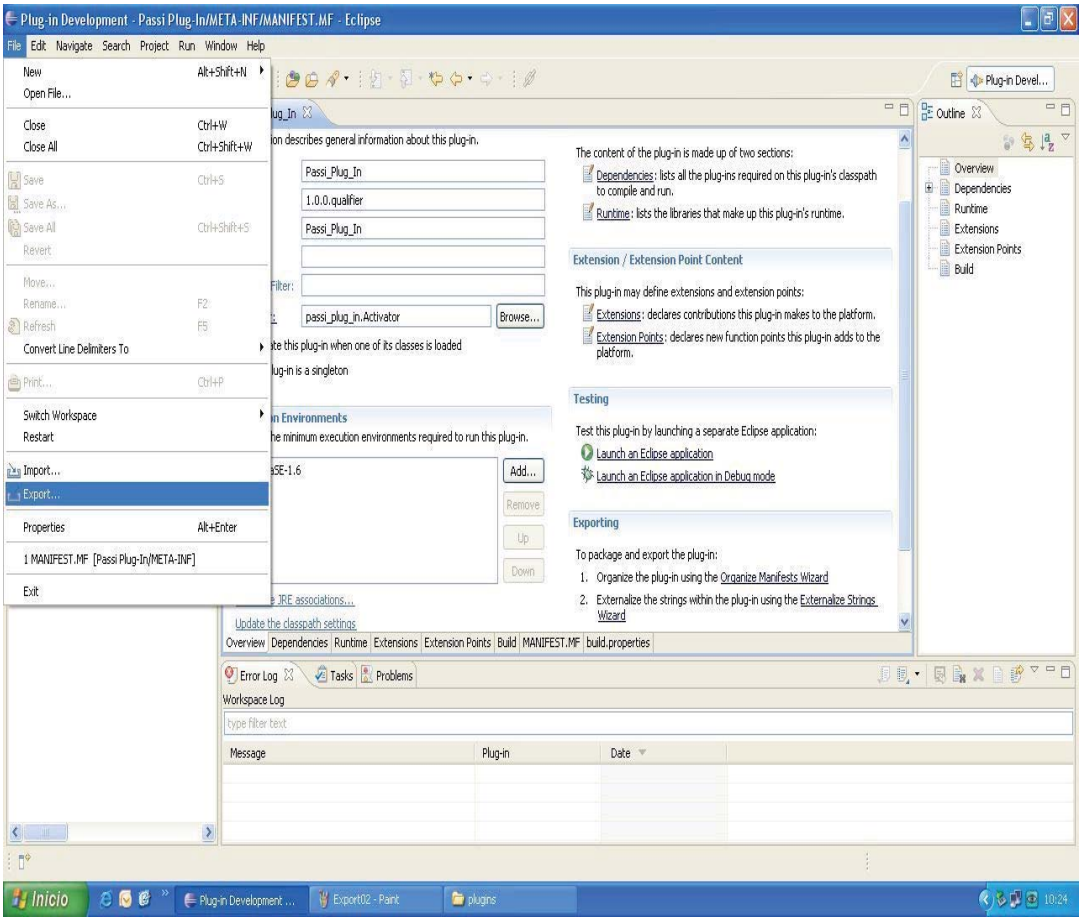

Figura 5.8: Export

2) En el cuadro de dialogo de exportación, seleccionar en la categoría "Java", la opción "Runnable JAR File" o "Archivo JAR ejecutable" (figura 5.9)

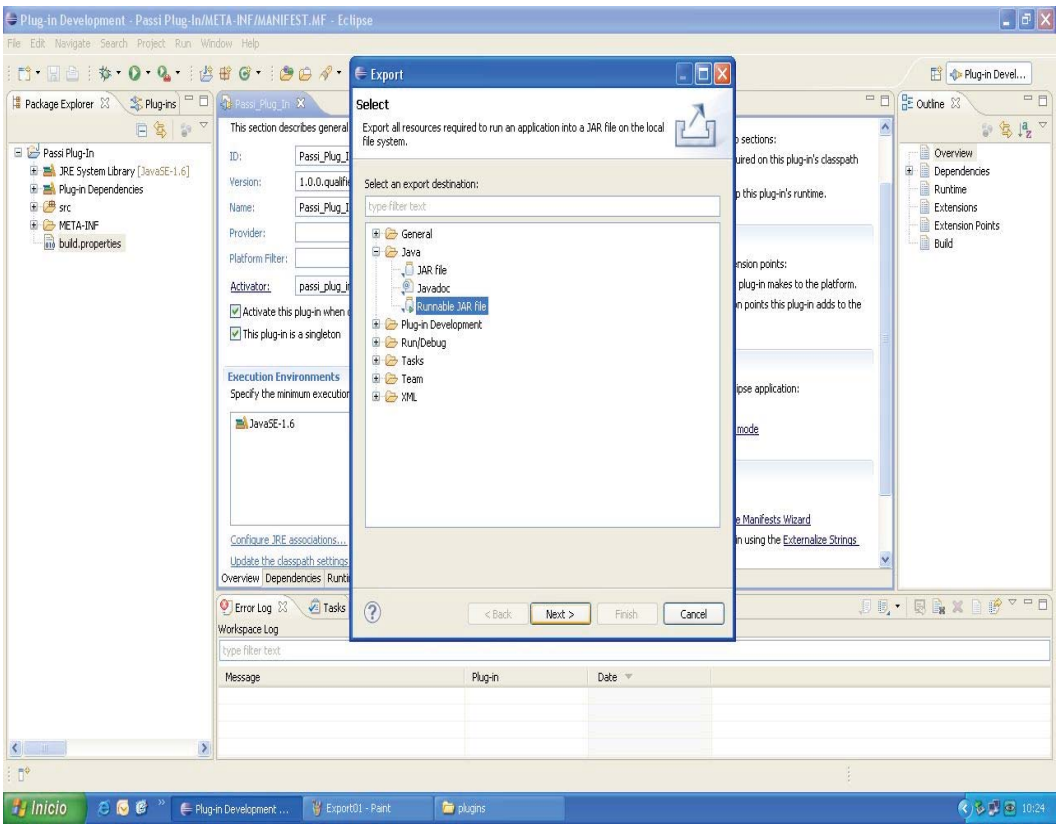

Figura 5.9: Select Runnable JAR File

3) En las opciones de exportacion de "Runnable JAR File" (figura 5.10), hay que seleccionar la configuración de lanzamiento (launch configuration), el nombre que tendra el archivo de exportación, y seleccionar la opción "Package required libraries into generated JAR". Presionar Finish para la generación del archivo .jar ejecutable en el destino previamente seleccionado.

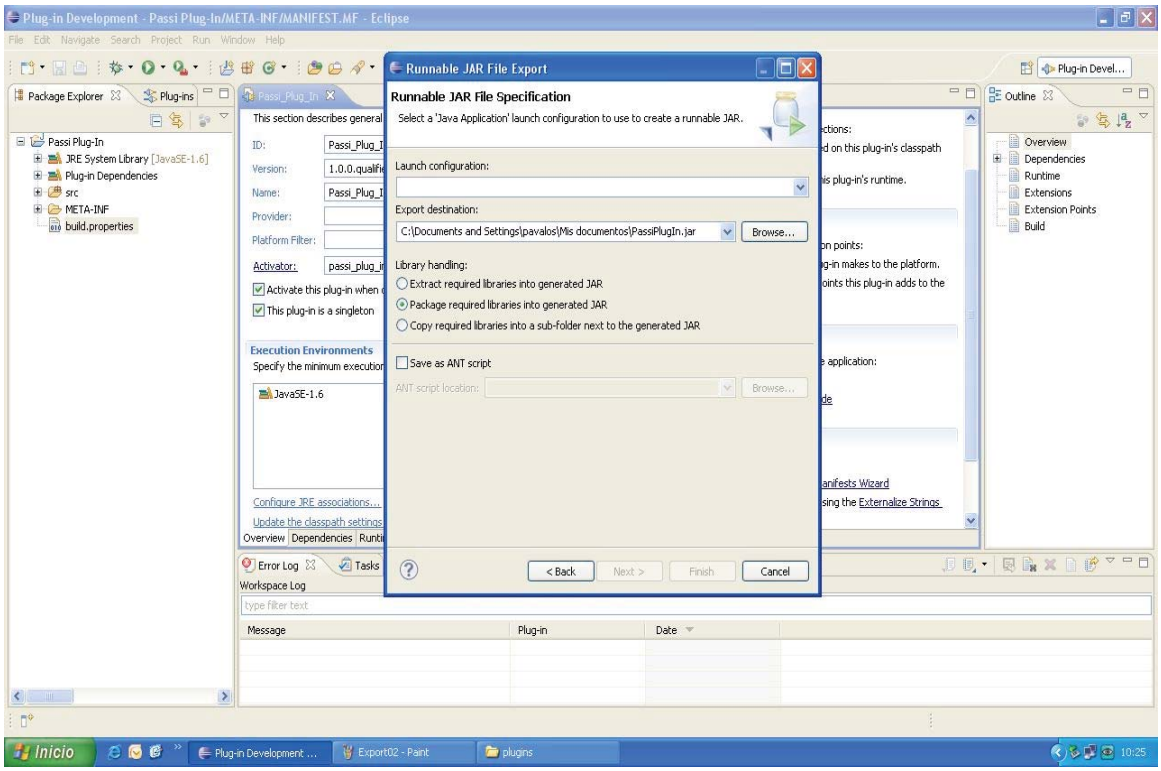

Figura 5.10: Jar File Export

Siguiendo estos pasos se obtiene el archivo .jar, el cual se puede utilizar como stand-along.

5.4.2 Fases Generadas

En la figura 5.11 se muestra la generación de un diagrama de Descripcion de Dominio. Este diagrama permite generar los casos de uso que mas tarde serviran para identificar los agentes y sus roles. Cabe mencionar que este diagrama permite al usuario acomodar los casos de uso sin que esta ubicación se pierda al cerrar la aplicación.

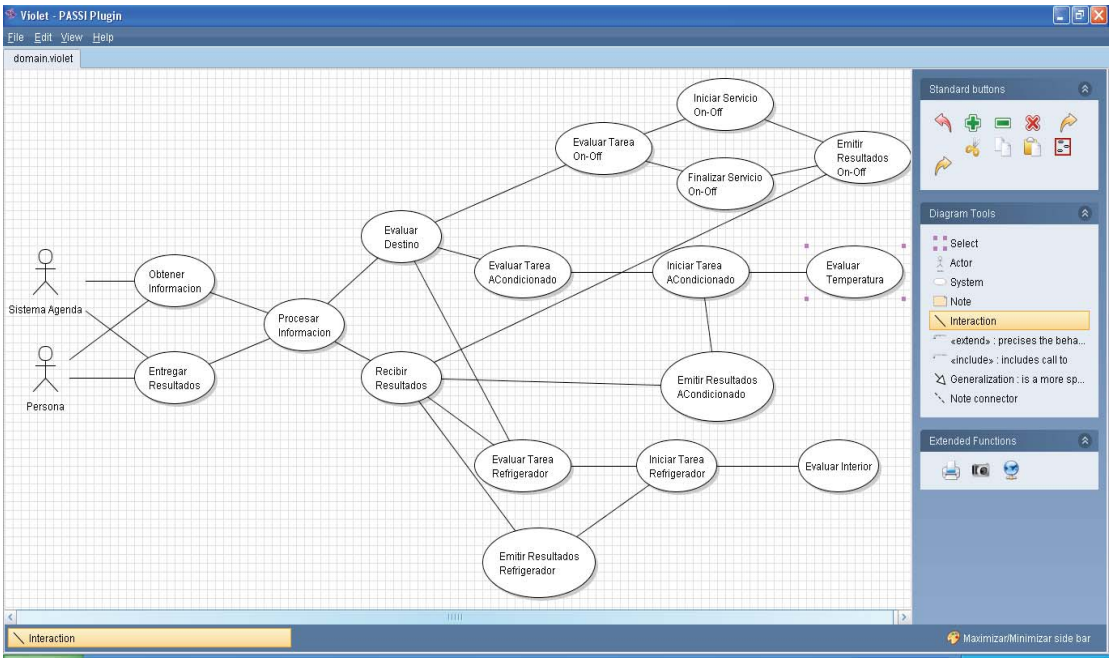

Figura 5.11: Captura Descripcion de Dominio

La figura 5.12 muestra la creación del diagrama de Identificación de Agentes, el que se genera a partir del modelo anterior empaquetando los casos de uso que corresponden a un determonado agente.

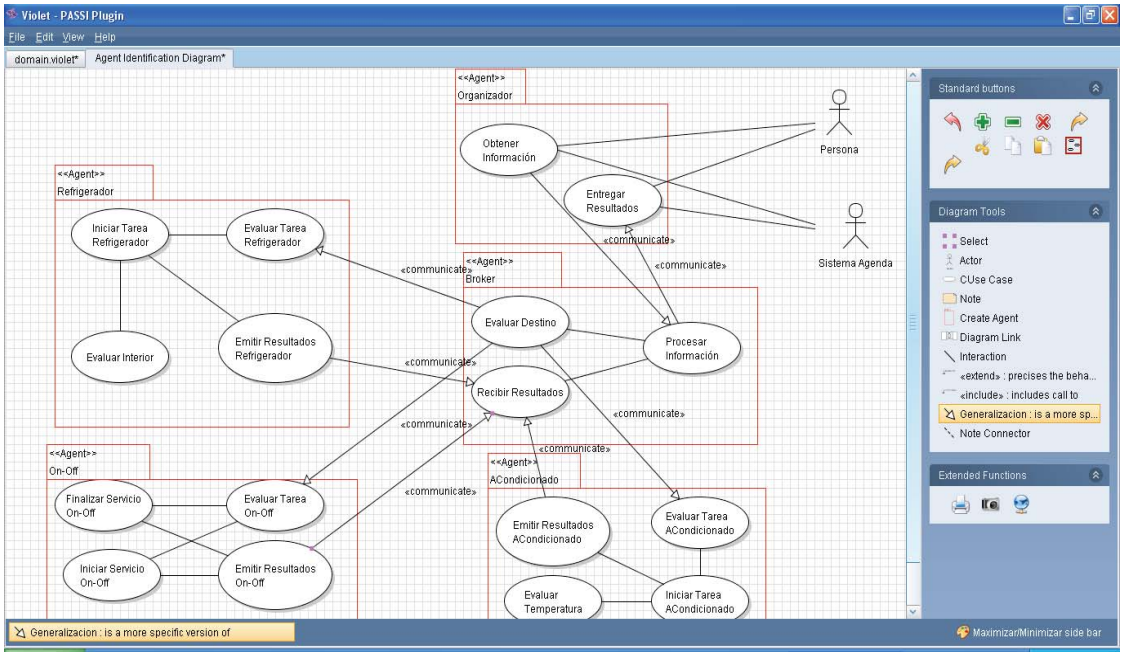

Figura 5.12: Captura Identificación de Agentes

En figura 5.13 se puede ver el diagrama de Identificación de Roles que permite asociar los roles que tomarán los agentes para la realizacion de las tareas.

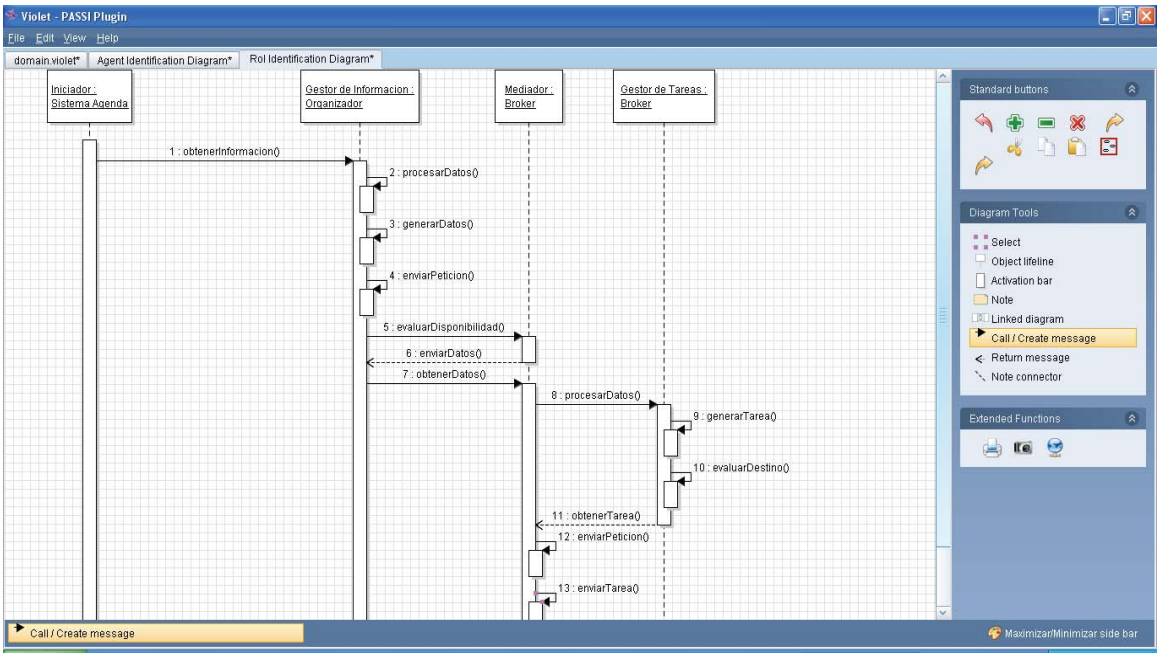

Figura 5.13: Captura Identificación de Agentes

En la figura 5.14 se muestra el diagrama de especificación de Tareas. Cabe destacar que los agentes aquí mencionados corresponden a los identificacdos en la figura 5.11.

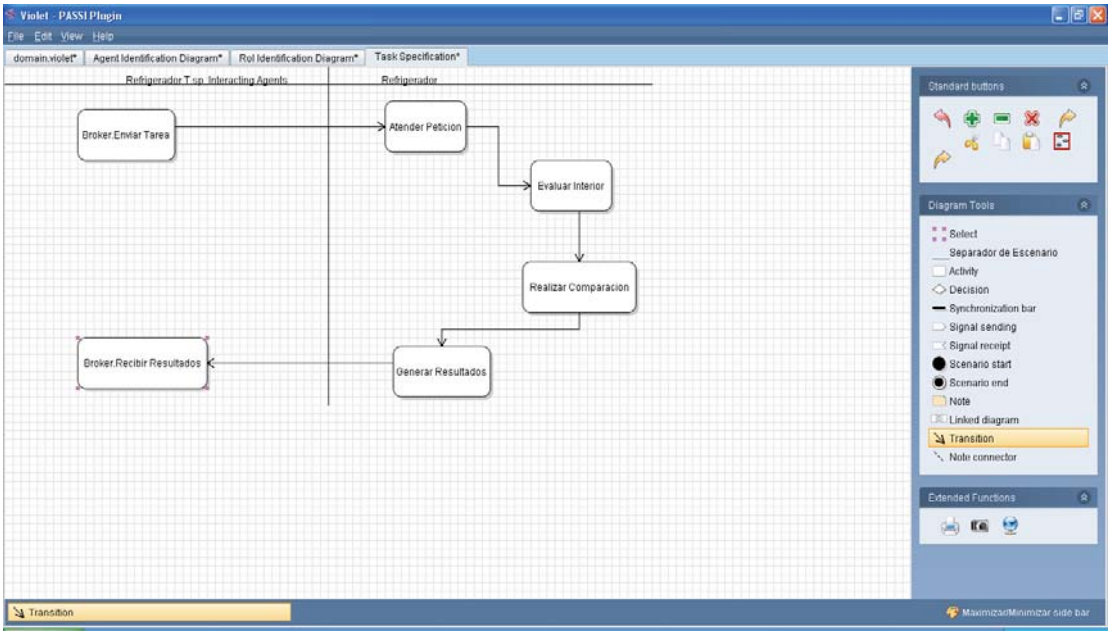

Figura 5.14: Captura Especificación de Tarea

5.5 Pruebas

Las pruebas realizadas son del tipo Caja Negra utilizando la técnica de Partición Equivalente. Este tipo de prueba consiste en agrupar los dominios de entrada del sistema de acuerdo a clases de datos comunes fin de obtener “clases” de errores. Para esto se agruparon los casos de uso que tienen datos de entrada y salidas esperadas similares para realizar pruebas grupales.

Para dichas pruebas se utilizaron diversos modelados, entre ellos: Kz-Domo Sistemas Multiagente para el Transporte de Pasajeros.

Las pruebas realizadas consistieron en realizar el modelado existente en la herramienta desarrollada y aplicar prueba y error, esto es, provocar errores intencionados a fin de probar el comportamiento de la herramienta. Se puede decir que la realización de estas pruebas fue altamente satisfactoria al cumplir con las expectativas planteadas. El documento de Plan de Pruebas se encuentra ubicado en el Anexo A de este informe.

6 Conclusiones

6.1 Conclusiones

Se lograron gran parte de los objetivos planteados al inicio de este proyecto, obteniendo un prototipo con la funcionalidad del modelo de Requerimientos del Sistema.

Cabe destacar que no existe mucha información en Internet acerca de la metodología PASSI que permita lograr una mayor claridad de las relaciones existentes entre los diagramas, sin embargo se han logrado determinar estas inter-relaciones.

Gracias al empleo de la metodología de Proceso Unificado, se simplificó el esfuerzo al momento de tener que realizar modificaciones al sistema. Esto se debió a que los desarrolladores contaban con experiencia en este campo.

En cuanto a la utilización del IDE seleccionado, fueron de muchas ayuda los foros de ayuda de Eclipse y la experiencia previa, aunque escasa, en su utilización. Eclipse demostró ser una herramienta sencilla y de fácil comprensión, aunque no presenta todos los recursos que puede llegar a ofrecer Netbeans, como una gran cantidad de tutoriales, herramientas de verificación del código, captación de errores y sus posibles causas, entre otras. Sin embargo, aunque no se contó con estos plus, se logro un manejo adecuado y una buena familiarización con Eclipse y Violet.

Se puede decir, a partir de estas conclusiones, que se cumplido con las expectativas iniciales de este proyecto.

6.2 Trabajo Futuro

El trabajo que queda pendiente depurar la herramienta desarrollada e implementar los modelos restantes de PASSI (Modelo de Implementación de Agentes, Modelo de Código y Modelo de Despliegue), para así lograr un modelado PASSI de inicio a fin con esta herramienta.

7 Referencias

[1] Pablo Cazau, “Introducción a la metodología de la investigación”, Buenos Aires, 2002

[2] Cossentino M, Potts C, “A CASE tool supported methodology for the desing of multi-agent system”, 2002

[3] Russell S, “Inteligencia Artificial: un enfoque moderno”, Pretince-Hall. Mexico, 1996

[4] Marchetti T, Garcia A, “Metodologias de Desarrollo de sistema multiagente: un análisis comparativo”,2001

[5] Burrafato J, Cossentino M, "Designing a multi-agent solution for a bookstore with t he PASSI methodology", 2002

[6] http://mozart.csai.unipa.it/ptk, “PTK 1.1 Tutorial”

[7] Cossentino M, "Different perspectives in designing multi-agent systems", 2002

[8] Wooldridge M, Ciancarini P. “Agent-Oriented Software Engineering: The State of the Art.”, 2001

[9] Cossentino, and Potts. “PASSI: a Process for Specifying and Implementing Multi-Agent Systems Using UML”

[10] Odell J, Van Dyke H, Bauer B. “Representing Agent Interaction Protocols in UML”,2001

[11] Jacobson I, Booch G, Rumbaugh J. “The Unified Process”, 1999

[12] Chella A., Cossentino M., Lo Faso U. “Designing agent-based systems with UML”, 2000

[13]Cossentino M, Sabatucci L, “Modeling Notation Source PASSI”, 2003

[14] www.csai.unipa.it/passi, documentación de PASSI

[15] Ian Somerville. Ingeniería de Software.

[16] Wooldridge M, Jennings N, “Intelligent Agents: Theory and Practice”, 1995

ANEXO A - Plan de Pruebas

A.1. Introducción

La construcción de un Plan de Pruebas es la piedra angular y en consecuencia el principal factor crítico de éxito para la puesta en práctica de un proceso de pruebas que permita entregar un software de mejor nivel. No obstante que cada esfuerzo o proceso de pruebas puede ser diferente y específico, la mayor parte de los proyectos informáticos, sean de nuevos desarrollos o de mantenimiento de aplicaciones, tienen un marco común para la realización de las pruebas.

El presente documento busca establecer el diseño de las pruebas para la Herramienta de Modelado PASSI y provistas por el grupo de desarrollo. El objetivo del plan de pruebas es encontrar la mayor cantidad de errores para garantizar la calidad del sistema que se entregará.

Una vez que la aplicacion se ha construido, es necesario hacerla pasar por una serie de ensayos antes de entrar a la fase de implementación. Mediante dichas pruebas, se medirá su reacción frente a diversas acciones que realizarán los usuarios desde sus páginas.

El plan de pruebas para una funcionalidad se compone de los supuestos que se considerarán como hechos para la ejecución de cada caso de prueba, además de la categorización de errores y el listado de todos los casos de pruebas contemplados en este plan.

A.2. Descripción de plan de pruebas

A.2.1. Aspectos Generales

A continuación se indican distintos aspectos que se deben considerar al momento de realizar un plan de pruebas.

A.2.2. Objetivo

El objetivo de la elaboracion del plan de pruebas es probar la funcionalidad de los módulos que componen la Herramienta de Modelado PASSI a fin de encontrar tempranamente los errores que la herramienta pueda presentar.

A.2.3. Entorno o marco

La aplicación a testear es una herramienta que mediante la generación de diagramas UML permite modelar un sistema multiagente bajo los parámetros de la metodologia PASSI.

Esta aplicación nace bajo la necesidad de automatizar la modelacion de los sistemas mltiagentes con PASSI. Mediante esta aplicación se podran generar cada uno de los diagramas exigidos por PASSI manteniendo la integridad de los mismos y permitiendo modificaciones que se vayan integrando al modelado sin generar inconsistencia.

A.2.4. Especificaciones del Sw y Hw

Con el fin de realizar adecuadamente las pruebas a la aplicación en cuestion es necesario contar con cierto equipamiento tanto de software como de hardware. Si bien los requerimientos son minimos y no se requieren de licencia alguna, existen ciertas especificaciones requeridas.

Software: La aplicación puede ser utilizada tanto como Stand-alone como pluging lo cual implica contar con el IDE Eclipse 5.2 version Europe (solo si se utiliza como plugin) ademas de tener instalado Java, este ultimo para ambas formas de ejecucion de la palicacion.

Hardware: Cualquier equipo minimo (128 de Ram, 1 giga de Disco duro) que soporte J2SE será suficiente para correr la aplicacion.

A.2.5. Alcance

La responsabilidad del diseño del plan de pruebas recae netamente en los desarrolladores del sistema, quienes se encargan de realizar el diseño de las pruebas para el sistema.

El plan a desarrollar requiere analizar la generacion y validacion de los diagramas que esta aplicación permite crear de modo que exista consistencia entre los diagramas.

A.2.6. Requisitos a probar.

Los requisitos que se llevarán a prueba serán todos los requisitos funcionales identificados en la Fase de Inicio y depurados en la en la Fase de Elaboracion, es decir todos los casos de uso agrupandolos por funcionalidades similares.

A.2.7. Estrategias de pruebas

Esta sección describe como el objetivo del plan será cumplido mediante la realización del siguiente tipo de prueba:

- Pruebas Caja Negra con el fin de ingresar ciertos datos validos de entrada y obtener un salida valida (causa-efecto), utilizando las técnicas de partición equivalente y análisis de valores limites.

- Técnica de Partición Equivalente: Se analizarán bajo el mismo parámetro aquellos Casos de Usa que tengan la misma funcionalidad y de quienes se espere la misma respuesta.

Estas pruebas serán diseñadas por el personal de desarrollo. Para estas estrategias de prueba definida deben detallarse los Casos de Prueba.

A.2.8. Casos de Prueba

Para realizar las pruebas a la Aplicación se tomará el sistema KZ-Domo como ejemplo y se recrearán sus diagramas para asi validar la aplicación y cada uno de sus modulos. Se forzarán errores de manera de comprobar la reaccion del sistema y poder detectar sus falencias.

A.3. Fichas de Pruebas

Cada caso de prueba consta de los siguientes elementos:

- Código-nombre: el código y nombre que definen al caso de prueba que se realizara.
- Objetivo: Reseña de lo que se busca lograr con el caso de prueba.
- Fecha de ejecución: se indica el día de ejecución de la prueba.
- Numero de toma: especificación del número de toma que se realiza para el caso de prueba.
- Numero de secuencia: Cantidad de veces que se ha realizado esta misma prueba.
- Ejecutores de la prueba: se especifica la(s) persona(s) que ejecutara(n) el caso de prueba.
- Testigos de la ejecución de la prueba: se indica la(s) persona(s) que será(n) testigo(s) en la ejecución del caso de prueba.
- Resultados de la prueba: se indica el resultado final que se obtuvo una vez que se realizo el caso de prueba, obteniendo como valores aprobada, aprobada con observaciones o fallida.
- Responsabilidad de aceptación: se detalla el nombre, fecha y firma de la(s) persona(s) responsables que aceptaran el caso de prueba una vez ejecutado.

- Descripción de los pasos: contempla el detalle de los pasos que componen el procedimiento de ejecución del caso de prueba, con su respectiva categorización del tipo de acción (Paso: fila blanca o Verificación: fila ploma) y el resultado obtenido para la acción (AP: aprobada, AO: aprobada con observaciones, FA: fallida, N/A: no aprobada).

- Observaciones: en este punto se puede exponer cualquier observación que sea necesario indicar y que se observo en alguna de las acciones realizadas en la ejecución del caso de prueba.

Las figuras A1 y A2 especifica cada elemento que se explicó anteriormente en el Detalle de las Fichas de Pruebas. La representación es de la siguiente manera:

DESCRIPCIÓN CASO DE PRUEBA

CÓDIGO – EB A E D

CODIGO PRUEBA– Titulo prueba **NOMBRE DEL CASO DE PRUEBA**

OBJETIVO

Rellenar con el objetivo de la prueba. Sirve también para definir condiciones de inicio y restricciones de la prueba.

FECHA DE EJECUCIÓN	**NÚMERO DE TOMA**	**NÚMERO DE SECUENCIA**

EJECUTORES DE LA PRUEBA	**TESTIGOS DE LA EJECUCIÓN DE LA PRUEBA**

RESULTADOS DPRUEBA (MARQUE CON UN ✓) **APROBADA:** ☐ **APROBADA CON OBSERVACIONES:** ☐ **FALLA:** ☐

RESPONSABILIDAD DE ACEPTACIÓN	**NOMBRE**	**FECHA**	**FIRMA**
JEFE DE PROYECTO			
RESPONSABLE			
USUARIO			
OTRO			

Figura A1: Ficha de Prueba Descripción

DESCRIPCIÓN DE LOS PASOS

CÓDIGO – EBA E D

CODIGO PRUEBA– Titulo prueba **Nombre del caso de prueba**				
A	**A**	**FA**	**N/**	**ACCIONES**
				1.
				1.
				2. En sombreado aquellos puntos que requieren verificación.
				4.
				5.

Observaciones:

Figura A2: Ficha de Prueba Pasos

A.3.1. Categorización de Errores

Para el desarrollo de cada caso de prueba, se utilizará el siguiente criterio para la categorización de los resultados que se puedan generar en cada acción de la descripción de pasos.

- Aprobada: la ejecución de la acción no arrojó errores resultando la realización de esta **satisfactoriamente.**
- Aprobada con observación: la ejecución de la acción arroja errores en alguna de las acciones, resultando la realización de esta con observaciones y posibles mejoras.
- Fallida: la ejecución de la acción arroja errores en alguna o todas las acciones o se realiza de manera inadecuada, resultando la realización de esta fallida.
- No Aprobada: la ejecución de la(s) acción(es) no realiza nada, resultando la realización de esta como no aprobada.

Y para la categorización del resultado final que se pueda generar una vez finalizada la ejecución del caso de prueba se utilizara el siguiente criterio:

- Aprobada: la ejecución del caso de prueba no arrojo errores resultando esta **satisfactoriamente y aprobada.**
- Aprobada con observación: la ejecución del caso de prueba arroja errores en alguna parte, resultando esta con observaciones y posibles mejoras.
- Falla: la ejecución del caso de prueba arroja errores o se realiza de manera inadecuada, resultando esta fallida.

A.4. Equipo de Pruebas

A continuación se indica las personas asociadas a la preparación y ejecución de las pruebas y su respectiva responsabilidad.

Nombre	**Cargo**	**Área**	**Responsabilidad**
Pablo Avalos Venegas	Estudiante Ingeniería Ejec. en informática	Desarrollo	Ejecución y diseño de pruebas
Deyanira Silva Urtubia	Estudiante Ingeniería Ejec. en informática	Desarrollo	Ejecución y diseño de pruebas

A.5. Incidencia de Errores

La clasificación que se utilizará para denotar el grado de incidencia de este efecto será:

- **Crítico**: Será una función o cambio que cause una falla general.
- **Severo**: Será una función o una interfaz que no actuará del modo planteado según los requerimientos.
- **Cosmético (no critico)**: Será la existencia de alguna mala ortografía en la interfaz o que hayan mensajes confusos que el usuario no comprenda.

ANEXO B - Manual de Usuario

B.1.Introducción

Este documento tiene por finalidad introducir al usuario en el funcionamiento de la Herramienta de Modelado para la metodología PASSI.

La Herramienta de Modelado para la metodología PASSI permite crear un modelado completo de las tres primeras etapas de PASSI, entregando una interfaz sencilla e intuitiva que permita al usuario modelar un sistema multiagente de forma rápida, íntegra y consistente.

B.2.Requerimientos

Esta aplicacion funciona en Windows, Linux y Mac y se puede utilizar como stand-alone o como un pluging de Eclipse Europe.

Para su correcta ejecucion se recomienda una memoria RAM mínima de 128 Mb y 500 Mb de disco duro.

B.3.Instalación

Para instalar la aplicación haga doble clic sobre el icono de la aplicacion y aparecerá la pantalla de bienvenida. Felicitaciones, ya se ha instalado la aplicación en su equipo.

B.4.Modelando Multiagente

Al inicio del programa (Figura B.1), debe escoger entre abrir uno de los archivos recientemente utilizados (columna derecha) o crear uno nuevo de los 7 tipos disponibles:

- Domain Description Diagram (DDD)
- Agent Identification Diagram (AID)
- Rol Identification Diagram (RIA)
- Task Specification Diagram (TSD)
- Domain Ontology Description Diagram (DODD)
- Comunication Ontology Description Diagram (CODD)
- Rol Description Diagram (RDD)
- Protocol Description Diagram (PDD)

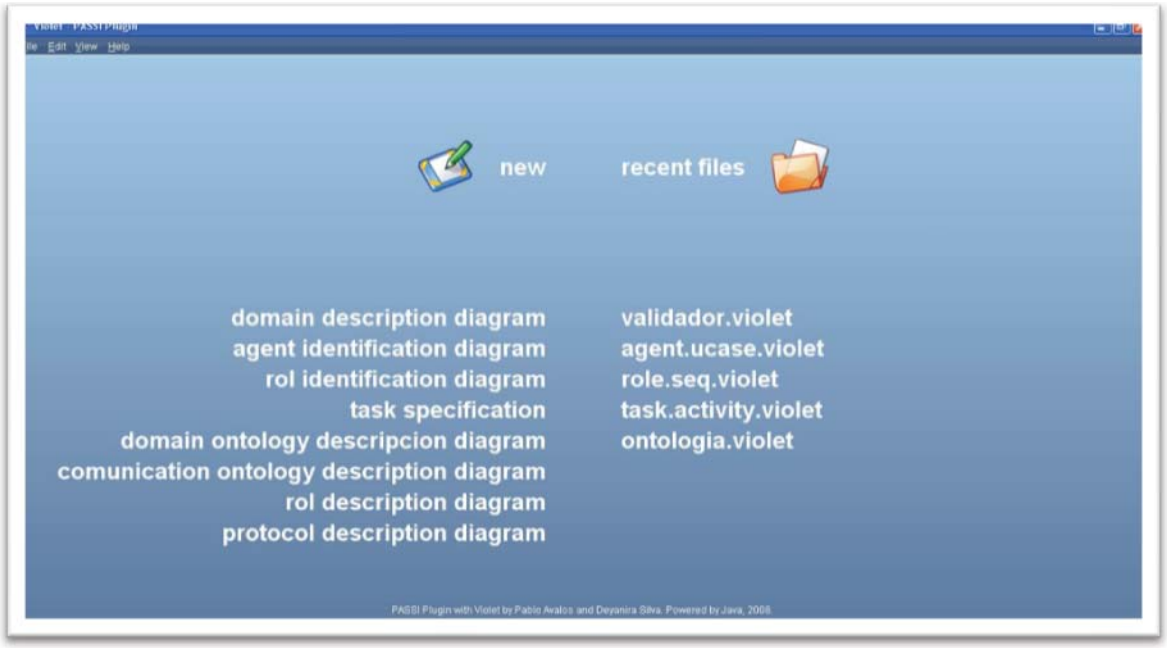

Figura B.1: Pantalla de Inicio

B.4.1. Funcionalidades Comunes

Cada uno de los siete diagramas disponibles para el usuario tiene un set de instrucciones en común (Figura B.2), denominado Standard o Standard Buttons, que funcionan de manera análoga a cualquier procesador de texto:

Figura B.2: Botones estandar

- Primera Fila: Deshacer Cambios, Zoom In, Zoom Out, Eliminar, Rehacer Cambios
- Segunda Fila: Cortar, Copiar, Pegar
- Tercera Fila: **Validar**

Esta última opción de validar le permite al usuario conocer el estado actual de integridad de su modelo, ya que permite detectar errores en la etapa actual y en las anteriores.

B.4.2. Domain Description Diagram:

El paso inicial en el modelado de Passi. El usuario encontrará en la seccion Diagram Tools, ubicada al costado derecho de la aplicación, un set de funcionalidades propias para cada diagrama; para el caso del Diagrama de Descripción de Dominio (DDD) son las siguientes (Figura B.3):

Figura B.3: Funcionalidades de DDD

- Select: Esta opción le permite al usuario seleccionar objetos ya existentes en el diagrama. Estos se pueden mover a una nueva posición manteniendo apretado el botón izquierdo del Mouse, o se puede acceder a las propiedades del objeto haciendo doble-click.

- Actor: Seleccionando esta opción, le permite al usuario ingresar nuevos actores al diagrama, por defecto el nombre es “Actor”, el cual se puede cambiar en las propiedades (doble click)

- System: La opción System permite ingresar un nuevo objeto de sistema al diagrama; no trae nombre por defecto, pero sí se puede ingresar uno en la ventana de propiedades.

- Note: La opción Note permite al usuario ingresar notas o memos dentro del diagrama. No tiene ningún tipo interacción con el diagrama en sí o los procesos de validación; sólo cumple la función ayuda-memoria.

- Interaction: Esta opción permite establecer las interacciones entre los distintos objetos del diagrama; una vez seleccionada, se debe mantener presionado el botón del ratón y arrastrar la línea de interacción hacia el objeto destino.

B.4.3. Agent Identification Diagram:

El segundo paso en el modelamiento de PASSI es el Diagrama de Identificación de Agentes (AID). Sus funcionalidades (figura B.4) son:

Figura B.4: Funcionalidades de AID

- Select: Esta opción le permite al usuario seleccionar objetos ya existentes en el diagrama. Estos se pueden mover a una nueva posición manteniendo apretado el botón izquierdo del Mouse, o se puede acceder a las propiedades del objeto haciendo doble-click.

- Actor: Seleccionando esta opción, le permite al usuario ingresar nuevos actores al diagrama, por defecto el nombre es “Actor”, el cual se puede cambiar en las propiedades (doble click)

- Use Case: Esta opción le permite al usuario ingresar nuevos casos de uso al diagrama, por defecto vienen sin nombre, el cual se puede modificar en la ventana de propiedades.

- Note: La opción Note permite al usuario ingresar notas o memos dentro del diagrama. No tiene ningún tipo interacción con el diagrama en sí o los procesos de validación; sólo cumple la función ayuda-memoria.

- Create Agent: Al seleccionar esta opción el usuario deberá generar un rectángulo que encierre los casos de uso que componen el agente. Por defecto su nombre es “Agente” y puede ser cambiado en la ventana de propiedades.

- Interaction: Esta opción permite establecer las interacciones entre los distintos objetos del diagrama; una vez seleccionada, se debe mantener presionado el botón del ratón y arrastrar la línea de interacción hacia el objeto destino.

B.4.4. Rol Identification Diagram:

El tercer paso en el modelamiento de PASSI es el Diagrama de Identificación de Roles (RID). Sus funcionalidades (figura B.5) son:

Figura B.5: Funcionalidades de RID

- Select: Esta opción le permite al usuario seleccionar objetos ya existentes en el diagrama. Estos se pueden mover a una nueva posición manteniendo apretado el botón izquierdo del Mouse, o se puede acceder a las propiedades del objeto haciendo doble-click.

- Object Lifeline: Seleccionando esta opción, le permite al usuario ingresar al diagrama nuevas “líneas de vida” de los objetos que desee

- Activation Bar: Estas barras de activación representan los procesos o actividades en los Object Lifeline, por lo que sólo pueden ser anexados a este tipo de objetos

- Call / Create message: Representa el inicio de la comunicación entre Activation Bars. Para crear una de estas llamadas, se debe arrastrar una línea desde el Activation Bar Origen al Destino (figura B.6).

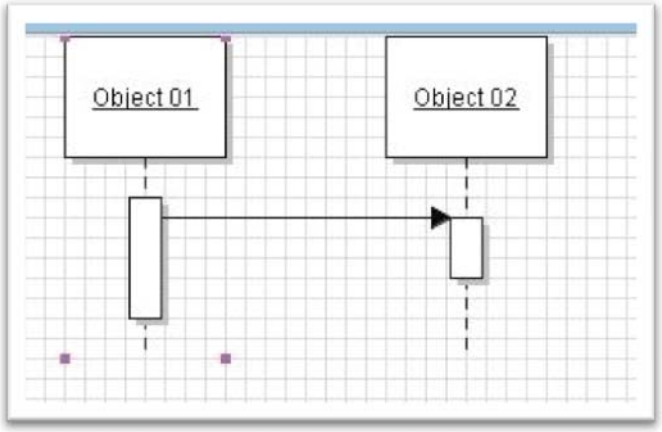

Figura B.6: Call/Create Message

Return Message: Representa la respuesta de la comunicación iniciada previamente con un “Call / Create Message”. Para crear una de estas respuestas, se debe arrastrar una línea desde el Activation Bar Origen al Destino (figura B.7).

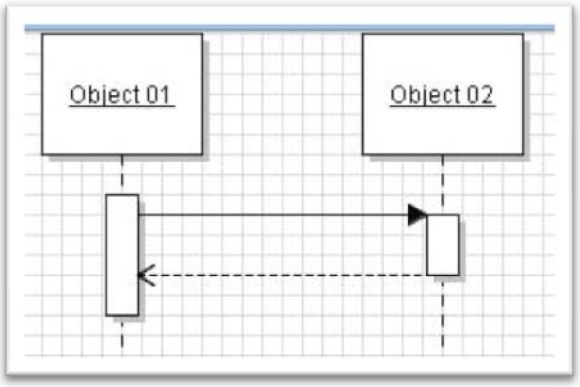

Figura B.7: Return Message

B.4.5. Task Specification Diagram:

El cuarto paso en el modelamiento de PASSI es el Diagrama de Especificación de Tareas (TSD). Sus funcionalidades (figura B.8) son:

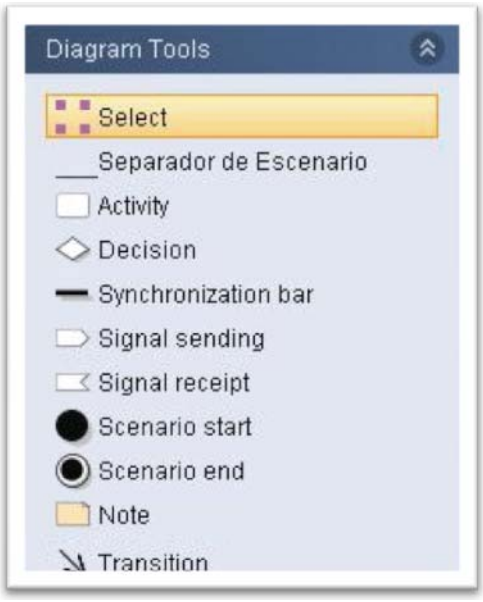

Figura B.8: Funcionalidades de TSD

- Separador: de escenario: divide las actividades relacionadas al agente a analizar del resto de los agentes que interactúan en el sistema
- Activity: describe la accion que se esta realizando (se describe con verbos como generar- validar- verificar)
- Decision: describe una decisión que genera un camino alternativo de actividades
- Syncronizacion bar: sirve para unir o dividir flujor de actividades
- Signal sending: representa el envío de una señal al finalizar un estado de acción
- Signal Receipt: representa la recepción de una señal al finalizar un estado de acción como entrada
- Scenario Start: inicio de escenario de actividades.
- Scenario end: fin de escenario de actividades.
- Note: La opción Note permite al usuario ingresar notas o memos dentro del diagrama. No tiene ningún tipo interacción con el diagrama en sí o los procesos de validación; sólo cumple la función ayuda-memoria.
- Transition: indica la transición por la que pasa la acción. Debe arrastrarse una línea desde el estado Destino al Origen.

B.4.6. Domain Ontology Description Diagram:

El quinto paso en el modelamiento de PASSI es el Diagrama de Descripcion de la Ontología de Dominio (DODD). Sus funcionalidades (figura B.9) son:

Figura B.9: Funcionalidades de DODD

- Select: Esta opción le permite al usuario seleccionar objetos ya existentes en el diagrama. Estos se pueden mover a una nueva posición manteniendo apretado el botón izquierdo del Mouse, o se puede acceder a las propiedades del objeto haciendo doble-click.

- Tipos de Clases: Se encuentran disponible para el usuario todas las clases necesarias para el modelado de passi en este diagrama; Action, Predicate y Concept. Cada una de ellas ingresa al diagrama solamente con el "tipo de clase", el resto de los detalles; nombre, metodos y atributos, deben ser ingresados en la ventana de propiedades.

B.4.7. Comunication Ontology Description Diagram:

El sexto paso en el modelamiento de PASSI es el Diagrama de Descripción de la Ontología de Comunicación (CODD). Sus funcionalidades (figura B.10) son:

Figura B.10: Funcionalidades de CODD

- Select: Esta opción le permite al usuario seleccionar objetos ya existentes en el diagrama. Estos se pueden mover a una nueva posición manteniendo apretado el botón izquierdo del Mouse, o se puede acceder a las propiedades del objeto haciendo doble-click.

- Actor: Seleccionando esta opción, le permite al usuario ingresar nuevas clases de agentes al diagrama, por defecto el nombre es "Agente", el cual se puede cambiar en las propiedades, así como agregar atributos y métodos

- Communication: Con esta opción se pueden agregar al sistema nuevas objetos de comunicación. En la ventana de propiedades se puede cambiar el nombre por defecto, así como editar los métodos que trae: Ontología, Lenguage y Protocolo

- Note: La opción Note permite al usuario ingresar notas o memos dentro del diagrama. No tiene ningún tipo interacción con el diagrama en sí o los procesos de validación; sólo cumple la función ayuda-memoria.

- Is Asociated with: Esta opción permite establecer las interacciones entre los distintos objetos del diagrama; una vez seleccionada, se debe mantener presionado el botón del ratón y arrastrar la línea de interacción hacia el objeto destino.

B.4.8. Rol Description Diagram:

El septimo paso en el modelamiento de PASSI es el Diagrama de Descripción de Roles (RDD). Sus funcionalidades (figura B.11) son:

Figura B.11: Funcionalidades de RDD

- Select: Esta opción le permite al usuario seleccionar objetos ya existentes en el diagrama. Estos se pueden mover a una nueva posición manteniendo apretado el botón izquierdo del Mouse, o se puede acceder a las propiedades del objeto haciendo doble-click.
- Rol: Permite agregar nuevas clases de Rol al diagrama, en la ventana de propiedades se pueden editar los detalles como Nombre, Métodos y Atributos
- Package Agent: Esta opción le permite al usuario ingresar un nuevo paquete de Agente en blanco, de manera que al interior de este pueda ir agregando los objetos de "Rol" que componen el paquete
- Rol Connection: Establece la relación de conexión entre los roles. Se debe arrastrar una línea desde el Rol de Origen al de Destino
- Rol Change: Establece la relación de cambio de rol entre los roles de un mismo paquete. Se debe arrastrar una línea desde el Rol de Origen al de Destino
- Note: La opción Note permite al usuario ingresar notas o memos dentro del diagrama. No tiene ningún tipo interacción con el diagrama en sí o los procesos de validación; sólo cumple la función ayuda-memoria.

B.4.9. Protocol Description Diagram:

El octavo paso en el modelamiento de PASSI es el Diagrama de Identificación de Protocolo (PDD). Sus funcionalidades (figura B.5) son:

Figura B.12: Funcionalidades de PDD

- Select: Esta opción le permite al usuario seleccionar objetos ya existentes en el diagrama. Estos se pueden mover a una nueva posición manteniendo apretado el botón izquierdo del Mouse, o se puede acceder a las propiedades del objeto haciendo doble-click.

- Object Lifeline: Seleccionando esta opción, le permite al usuario ingresar al diagrama nuevas “líneas de vida” de los objetos que desee

- Activation Bar: Estas barras de activación representan los procesos o actividades en los Object Lifeline, por lo que sólo pueden ser anexados a este tipo de objetos

- Call / Create message: Representa el inicio de la comunicación entre Activation Bars. Para crear una de estas llamadas, se debe arrastrar una línea desde el Activation Bar Origen al Destino.

- Return Message: Representa la respuesta de la comunicación iniciada previamente con un “Call / Create Message”. Para crear una de estas respuestas, se debe arrastrar una línea desde el Activation Bar Origen al Destino.

B.5.Nota

Ante alguna duda sobre el funcionamiento de esta herramienta, puede enviar un correo a deyanirasilvau@gmail.com con asunto “Herramienta PASSI” y detallando su consulta. Recibira una respuesta en un plazo entre 2 y 10 días.