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

VISUALIZACION 3D DE SISTEMA MULTIAGENTE APLICADO AL PROBLEMA DE TRANSPORTE DE PASAJEROS

FELIPE ANDRÉS BARANLLONI LAGOS

TESIS DE GRADO
MAGÍSTER EN INGENIERÍA INFORMÁTICA

DICIEMBRE 2009

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

VISUALIZACION 3D DE SISTEMA MULTIAGENTE APLICADO AL PROBLEMA DE TRANSPORTE DE PASAJEROS

FELIPE ANDRÉS BARANLLONI LAGOS

Profesor Guía: **Claudio Cubillos Figueroa**

Programa: **Magíster en Ingeniería Informática**

Diciembre 2009

Resumen

Un Sistema Multiagente es un conjunto de programas autónomos inteligentes programados para resolver un problema determinado, cooperando entre sí, intercambiando mensajes, etc. Este enfoque de cooperación sería mucho más fácil de visualizar mediante algún tipo de interfaz gráfica que permitiese ver esta interacción explícitamente. Para ello se pretende realizar una interfaz en 3D donde cada agente tenga una representación propia en este mundo 3D, y el intercambio de mensajes se pueda apreciar claramente. Para todo esto se seleccionará un motor gráfico 3D desarrollado en java de preferencia, puesto que el framework con el cual se desarrolla el Sistema Multiagente a graficar también está programado en el mismo lenguaje. Los modelos 3D de cada agente y objeto a graficar serán modelados con una herramienta especializada externa, y luego importados al motor gráfico Java.

Palabras Clave: Sistema Multiagente, JMonkeyEngine, Gráficas 3D.

Summary

A multiagent system is a set of intelligent autonomous programs developed to solve a particular problem, cooperating with each other, exchanging messages, etc. This cooperative approach would be much easier to see through some kind of graphical interface to see this interaction explicitly. For this, a 3D interface is developed where each agent has its own representation in the 3D world, and the exchange of messages can be clearly seen. A 3D graphical engine will be selected, developed preferively in java, since the framework with which the multiagent system to graph is developed is also in the same language. The 3D models of each agent and object to draw are modeled with an external specialized tool and then imported into the Java graphical engine.

Key Words: Multiagent System, JMonkeyEngine, 3D Graphics.

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1 DESCRIPCIÓN DE SISTEMA

1.1 Introducción

La utilización de agentes para la resolución de problemas últimamente ha comenzado a tomar relevancia, debido al nivel de complejidad de los problemas que pueden ser resueltos por ellos. Otra característica interesante radica en su facilidad para poder realizar tareas descentralizadas, debido a ello es que su uso es cada vez más estudiado.

Junto con ello existen sistemas en los que se encuentran involucrados varios agentes en conjunto para poder dar solución a una situación compleja, ya sea un sistema de distribución de carga en un puerto o un sistema de transportes. Este tipo de sistemas en los que varios agentes están involucrados en la resolución de un problema o domino específico, son conocidos como Sistemas Multiagente (representados por sus siglas en inglés M.A.S.). Sin embargo la mayoría de los MAS sólo poseen una que otra interfaz de control donde se ve una lista de agentes y contenedores. Esto hace difícil la comprensión de los mismos u observar cómo se está llevando la comunicación entre los distintos agentes involucrados.

Por esto es que en el siguiente proyecto se propone la realización de una interfaz en 3 dimensiones donde poder ver una representación gráfica de cada uno de los agentes, y observar claramente la interacción entre ellos, visualizando los mensajes que intercambian, ayudando a la comprensión tanto por profesionales, como con fines meramente educacionales.

Al realizar el desarrollo de esta interfaz, se busca que funcione lo más transparente posible para el Sistema Multiagente seleccionado. Con esto se busca que las modificaciones necesarias que deba sufrir el Sistema sean mínimas, y en caso óptimo inexistentes.

1.2 Definición de Objetivos

1.2.1 Objetivo General

Desarrollar una interfaz gráfica 3D para visualizar el comportamiento de un Sistema Multiagente para transporte de pasajeros.

1.2.2 Objetivos Específicos

- Analizar cómo funciona un sistema Multiagente y su plataforma de desarrollo, y determinar cuál es la mejor forma de sincronizarlo con la interfaz 3D a desarrollar.

- Determinar el mejor motor gráfico a utilizar para el desarrollo de la interfaz 3D

- Desarrollar los diferentes modelos 3D requeridos por la interfaz e integrarlos con la misma.

- Realizar la integración de la Interfaz 3D con un Sistema Multiagente para transporte de pasajeros.

- Investigar y establecer la forma de validar el correcto funcionamiento de la interfaz 3D.

1.3 Plan de Trabajo

El siguiente proyecto será realizado utilizando la metodología UP, debido a que permite llevar a cabo una planificación del proyecto mucho más detallada y específica, permitiendo controlar los procesos del proyecto, además de proveer herramientas con las cuales se puede ser más ordenado respectos a las tareas que se deben realizar en él. En la Ilustración 1-1 se presenta la carta Gantt asociada al proyecto.

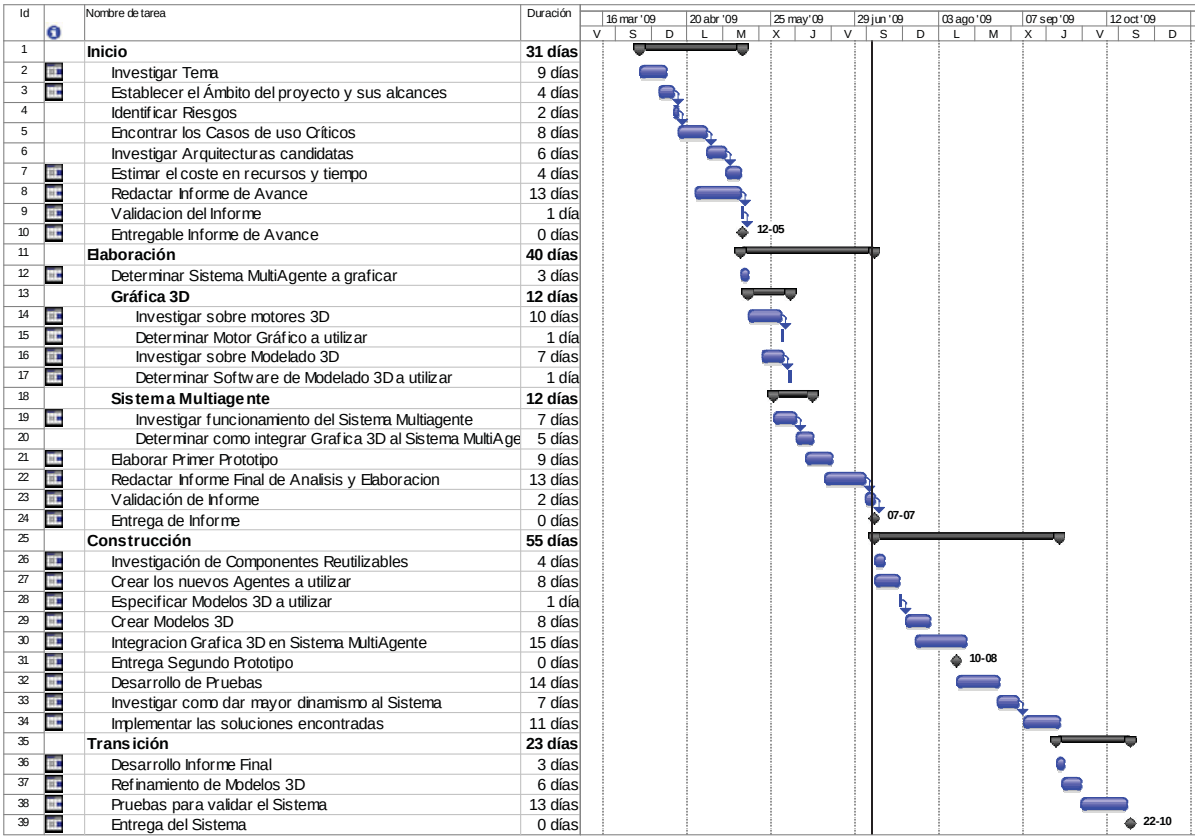

Ilustración 1-1 Plan de Trabajo

1.4 Metodología

Para la realización de este proyecto se consideraron las diferentes metodologías para llevarlo a cabo, entre las cuales se encuentran Proceso Unificado (UP), Modelo con prototipos evolutivos, y Cascada. Finalmente la seleccionada fue UP. Entre sus características encontramos que se realiza el trabajo en forma incremental y además utiliza UML en lo que se refiere al análisis y captación de requerimientos, esto facilita la tarea referida a todo lo que se refiere a los primeros pasos dentro del desarrollo del software. Además este modelo se enfoca a la calidad, lo que proporciona un marco de trabajo en el cual no sólo se estará elaborando la solución de nuestro sistema sino que

también se está asegurando que el producto final cumpla con lo requerido. Además el elaborar todo el software guiado por los casos de uso permite el dividir tareas de forma sencilla. Además brinda las ventajas asociadas a un proceso incremental e iterativo. Con esto se puede observar cómo se avanza en la creación del software debido a que brinda una serie de herramientas visuales con las cuales vigilar el avance del proyecto. Una ventaja adicional que entrega UP es el hecho de que sea incremental, puesto que permite dar una mayor cantidad de tiempo a la investigación del tema y brinda la oportunidad de tomar las decisiones que permitan realizar el proyecto de manera correcta.

1.5 Estructura del Documento

El Documento está organizado como sigue:

- El Segundo introduce al paradigma de agentes, presentando los sistemas multiagente con las herramientas y tecnologías asociadas a su desarrollo e implementación.

- El Tercero da una reseña sobre qué son los motores gráficos 3D, para qué se utilizan, y algunas técnicas que utilizan junto a una lista de algunos de ellos.

- El Cuarto capítulo habla sobre los modelos 3D, qué son, cómo se realizan, y qué técnicas de modelado existen.

- El Quinto nos da una descripción del sistema multiagente seleccionado para visualizar, así también el cómo funciona y su uso.

- El Sexto habla del desarrollo del Sistema tratado en este proyecto, cómo se pretende realizar esto, y cual el trabajo realizado.

- El Séptimo habla sobre las pruebas a realizar, describiendo la configuración actual.

- Finalmente el Octavo capítulo presenta las conclusiones que se obtuvieron al realizar el trabajo.

2 SISTEMAS MULTIAGENTE

A lo largo del desarrollo de este trabajo se habla continuamente de los Sistemas Multiagente, en este capítulo se trata todo lo referente a ellos, como es una descripción del concepto de agente y sistema multiagente, sus diferentes arquitecturas y metodologías de desarrollo, además de cuáles son las diferentes alternativas de plataformas de desarrollo existentes.

2.1 Agentes

La tecnología no para en su avance y cada cierto tiempo comienzan a surgir propuestas que nos llevan un paso más en el progreso tecnológico. Así después de la revolución de la orientación a objetos, una nueva promesa irrumpe con fuerza, los agentes. Pero qué entendemos por este término, numerosos investigadores se han dedicado a resolver esta pregunta, consideremos algunas definiciones que se han dado con respecto al término agente por estudiosos del tema:

"Toda entidad que de forma autónoma perciba su entorno (real o simulado) mediante sensores y que actúe en el mismo mediante efectores" [Russell y Norvig, 2003].

Según la definición anterior un agente, actúa en forma autónoma, y puede interactuar con su entorno, sin embargo, esta definición es demasiado general y no nos permite distinguir un agente de otros términos. Consideremos entonces esta otra definición:

"Un agente es un sistema informático que se sitúa en un cierto ambiente y que es capaz de acción autónoma en este ambiente para lograr sus objetivos de diseño" [Wooldridge y Jennings, 1995]

Esta definición nos extiende a que además de ser autónomos e interaccionar con su entorno, los agentes realizan interacciones para lograr sus objetivos o tareas específicas, pero cómo lograr esto, cómo razonar sobre qué acciones llevar a cabo y de qué manera interactuar con su entorno. Vuelve entonces a surgir otra definición que abarca esta área:

"Los agentes inteligentes continuamente realizan tres funciones: percepción de condiciones dinámicas en el entorno; acciones para afectar las condiciones en el entorno; y el razonamiento para interpretar percepciones, resolver problemas, sacar conclusiones, y determinar acciones." [Hayes-Roth, 1995]

Así como estas definiciones se pueden encontrar un sin fin de ellas, relacionadas a distintos aspectos, como movilidad, comunicación, flexibilidad, etc. Todas estas definiciones comparten el hecho de que son realizadas desde el punto de vista del área desde la cual fueron propuestas. Viéndolo de esa forma, la última definición es la que nos resulta más útil debido al hecho de que unifica todos los puntos de vista. Sin embargo podemos utilizar lo que dicen [Wooldridge y Jennings, 1995] para distinguir dos nociones de agentes. La primera es conocida como "noción débil de agente", la cual agrupa las características mínimas que debiese tener cualquier agente:

- Autonomía, un agente encapsula un estado propio, pudiendo actuar sin intervención directa de terceros, ya sean humanos o otros sistemas. Un agente controla sus comportamientos, y por ello puede tomar las decisiones que crea convenientes para sí mismo.

- Sociabilidad, los agentes pueden interaccionar con otros agentes o humanos utilizando alguna especie de lenguaje de comunicación de agentes. De este modo, pueden intercambiar conocimientos, pueden negociar, o pueden cooperar para resolver un problema.

- Reactividad, Un agente está inmerso en un determinado entorno, el cual puede ser físico (constituidos por objetos del mundo real) o virtual (constituido por sistemas de agentes, otros sistemas, Internet, etc.), del que percibe estímulos y ante los que debe reaccionar en un tiempo preestablecido.

- Pro-actividad, los agentes no sólo deben reaccionar a los cambios del entorno, sino también deben poseer un carácter emprendedor que le permita tomar la iniciativa para cumplir con sus objetivos o tareas.

Esta noción de agente es generalmente utilizada por la ingeniería de software orientada a agentes (AOSE, Agent Oriented Software Engineering); la segunda definición es conocida como "noción fuerte de agente" y es utilizada comúnmente por la inteligencia artificial, en esta noción se incluyen las características de un agente débil y agrega las siguientes definiciones:

- Adaptabilidad, los agentes no sólo reaccionan frente a los cambios del ambiente. También están en un continuo proceso de adaptación a los cambios, lo que les permite mantenerse viables a lo largo del tiempo.

- Movilidad, Es la capacidad del agente para moverse de forma autónoma en una red, es decir, el agente debe ser capaz de suspender su ejecución en un servidor y reanudarla en otro servidor una vez que se haya desplazado a éste, manteniendo los datos que posee.

- Veracidad, la información transmitida por un agente es veraz.

- Racionalidad, Se asume que el agente intentará alcanzar las metas que le fueron asignadas de la mejor forma posible, basándose en la información que posee del mundo exterior.

2.2 Sistema Multiagente

La mayoría de las veces los agentes son capaces de trabajar de manera independiente interactuando con otros agentes cuando requieren información, esta forma es usualmente la más implementada [Hayes-Roth, 1995]. Sobre esta base de interacciones se pueden construir una sociedad de agentes, que simula a las organizaciones humanas, en las cuales un grupo de individuos trabaja en forma conjunta para alcanzar las metas propuestas. De esta manera, al igual que una organización humana, un MAS (Multi Agent System) debe tratar con los problemas intentando resolverlos utilizando los recursos disponibles de la manera más eficientemente posible. De esta manera nos encontramos con la siguiente definición para un MAS:

"Conjunto de solucionadores de problemas o agentes, que interactúan entre sí para encontrar respuestas a problemas que van más allá de sus capacidades o conocimientos individuales de cada entidad". [Weiss, 1999]

De este modo, la investigación en el campo de los MAS está principalmente orientada a coordinar los comportamientos entre los agentes que interactúan. Esto quiere decir, establecer los medios por los cuales pueden compartir conocimiento, metas, habilidades y planes. De este modo, es necesario lograr coherencia entre los comportamientos del sistema de agentes entero. Sin embargo, lograr esta coherencia no es fácil, debido a que no hay un control global de los participantes, además debe considerarse el hecho de que pueden establecer interacciones de modos no predefinidos.

Para lograr coordinación dentro de un MAS, en [Weiss, 1999], se identifican 2 medios por los cuales lograr esto: cooperación, y competición. No obstante, detrás de ambos conceptos se identifican las mismas necesidades:

- Las dependencias entre las acciones de los agentes, cuando los objetivos asignados a agentes individuales están relacionados.

- No hay necesidad de restricciones globales.

- Los agentes individualmente no tienen suficiente competencia, recursos o información para resolver el problema entero.

Sin embargo, las mismas propiedades que vuelven el desarrollo de estos sistemas una acción complicada, son las que les confieren ventajas y fortalezas como producto de software.

- Su naturaleza distribuida le confiere la capacidad de expandir las tareas a distintos recursos de un mismo entorno, logrando de esta forma aumentar la productividad.
- Si se utilizan adecuados protocolos para la interacción entre los agentes, el sistema resultante puede ser tolerante a fallos y modificable en tiempo real.

- El hecho de simular una estructura organizacional de personas resulta en que el diseño de alto nivel del sistema puede resultar más intuitivo para los desarrolladores.

2.3 Arquitecturas para Sistemas de Agentes

En esta sección se describen las principales características de los sistemas de agentes, considerando la arquitectura interna, sus interacciones con otros agentes y su movilidad.

2.3.1 Arquitectura de un Agente

Un agente es una entidad que realiza acciones como consecuencia de cómo percibe el entorno sobre el cual se desarrolla. Este proceso puede ser descrito desde 2 puntos de vista: abstracto y físico.

2.3.1.1 Arquitecturas de Agentes Abstractas

Considerando que los agentes pueden reaccionar de distinto modo de acuerdo a sus comportamientos definidos**[Weiss, 1999]**, se han establecido los siguientes tipos:

- Agentes simplemente reflectantes (o agentes puramente reactivos), son los agentes más simples, están limitados sólo a tomar decisiones de acuerdo a la percepción que tienen de su entorno, sin considerar un historial de acciones.

- Agentes basados en Modelos (o agentes con estado), estos agentes mantienen un historial interno (o una parte de él), el cual les permite decidir qué acciones realizar en el entorno.

- Agentes Basados en Objetivos, estos agentes aplican un orden a los objetivos, para de esta forma evaluar qué acciones realizar, después genera las posibles acciones a realizar basado en las percepciones del entorno. Comúnmente las técnicas de IA que dan soporte al logro de objetivos están relacionadas a la búsqueda y planeación.
- Agentes Basados en la utilidad, estos agentes poseen una función de utilidad que les permite discriminar entre distintas formas de lograr el mismo objetivo. Esto se alcanza introduciendo una variable que permita evaluar el plan en términos de costo, fiabilidad, tiempo, utilidad, etc.

- Agentes que aprenden, estos agentes tienen la capacidad de aprender sobre sus propias acciones. Ellos tienen un elemento de aprendizaje, el cual está encargado de realizar mejoras en los comportamientos del agente, y un elemento realizador, el cual tiene por acción seleccionar las acciones externas al agente. El elemento de aprendizaje utiliza retroalimentación desde un elemento crítico, sobre cómo el agente lo está haciendo (comparando sus acciones con un funcionamiento estándar fijo), y determinando cómo modificar el elemento realizador para mejorar el resultado en el futuro. Estos agentes también pueden tener un elemento solucionador de problemas el cual es responsable de la generación de nuevos casos que pueden producir nueva información de experiencia.

2.3.1.2 Arquitecturas de Agentes Concretas

Las arquitecturas concretas implementan los modelos abstractos descritos anteriormente, describiendo la estructura interna y las operaciones de los agentes, en [Weiss, 1999] se han definido las siguientes arquitecturas:

- Deliberativas: esta categoría de agentes es implementada usando las aproximaciones tradicionales para construir sistemas inteligentes en el dominio de la IA. El agente maneja una representación simbólica de su entorno y comportamientos, esto le permite manejarlos en forma sintáctica. Un agente basado en la lógica determina sus comportamientos por medio de la aplicación de ciertas reglas de deducción, entregadas por el programador, sobre el conocimiento base que contiene los hechos o información que el agente tiene sobre su entorno. La principal desventaja de esta arquitectura deliberativa está relacionada con la ausencia de garantías de que el procedimiento para la toma de decisiones termine de manera oportuna, y la dificultad que puede ser encontrada al intentar llevar un entorno a una representación simbólica (el cual es un problema aún abierto en el campo de la IA).

- Reactivas: esta arquitectura fue propuesta para producir agentes capaces de actuar en entornos con restricciones de tiempo. El punto central de esta arquitectura es considerar los agentes como entidades que tienen un conjunto de comportamientos básicos, las cuales son simplemente el resultado de la

interacción que el agente mantiene con su entorno. De esta forma, los comportamientos inteligentes surgen de la interacción de los comportamientos básicos, los cuales son expresados como maneras de trazar situaciones en el entorno. Una importante característica es que ni la representación simbólica ni el razonamiento simbólico son adoptados en este modelo. Inclusive, estos agentes no implementan ningún modelo de su entorno, esto se debe a que solamente requieren información suficiente para su estado local (su estado actual). La principal desventaja de esta arquitectura aparece debido a su simplicidad: primero, la información que no es propia del agente no es considerada, por ello los agentes sólo pueden tomar decisiones de poco alcance; segundo, estos agentes no son capaces de aprender (y mejorar su rendimiento) desde su experiencia; y tercero, no hay una metodología existente para construirlos (por eso ellos deben ser desarrollados de manera experimental).

- Híbridas: debido a que ninguna de las arquitecturas anteriores han demostrado ser óptima para la construcción de agentes, se han planteado soluciones de arquitectura híbridas en las cuales se mezclan ambos métodos. Una propuesta se basa en la construcción de dos subsistemas: uno deliberativo en el cual planes y decisiones son planteadas con un conjunto de símbolos; y otro reactivo el cual tenga la capacidad de reaccionar frente a eventos del entorno sin utilizar un razonamiento complejo. Esta metodología se puede desarrollar en capas en la cual una o más de estas capas pueden tener acceso a los datos suministrados por el entorno y una o más capas pueden realizar acciones en el entorno. A partir de esto se presentan 2 clases de arquitectura híbrida que pueden ser desarrolladas:

  - Horizontal: todas las capas tienen acceso a los datos del entorno y a realizar acciones en él.

  - Vertical: sólo una capa tienen acceso a los datos y a realizar acciones en el entorno.

Debido a esta arquitectura en capas, el correcto comportamiento de los agentes viene dado por la interacción entre los diferentes niveles y en el nivel de información de cada capa.

- Arquitectura BDI (Belief-Desire-Intention, en español Creencias, Deseos e Intenciones): la arquitectura BDI tiene sus bases en la aproximación filosófica del razonamiento práctico. El razonamiento práctico divide los comportamientos en 2 procesos: deliberación (decidiendo las metas a alcanzar) y razonamiento medios-fin (decidiendo como las metas serán logradas). Estos procesos son soportados por tres componentes básicos de la arquitectura BDI, que corresponden a estructuras de datos que representan las creencias, deseos e intenciones del agente. Creencias son la información que el agente conoce sobre su entorno y sobre sí mismo. Deseos son los objetivos o motivos que caracterizan la meta que el agente desea lograr. Usualmente, un agente BDI también es llamado conocimiento procedimental

debido a que tiene un conjunto de planes para alcanzar el objetivo o para reaccionar a ciertas situaciones. Las intenciones representan estados deliberativos, el cual está constituido por una las opciones seleccionadas. Como desventaja de este modelo, [Georgeff et al., 1998] cita que es inadecuado para la construcción de sistemas que deben aprender y adaptar sus comportamientos.

2.3.2 Arquitectura Multiagente

Una de las arquitecturas más aceptadas y utilizadas actualmente es la arquitectura FIPA **[FIPA, 2002]** , la mayoría de los entornos de desarrollo y ejecución de agentes tiende a ser compatibles con esta arquitectura, esto se debe a que en sus especificaciones se definen una serie de características que deben cumplir las plataformas de gestión para sistemas multiagente, de esta forma se busca tener una colección de estándares que promuevan la interoperabilidad de agentes heterogéneos y de los servicios que representan.

FIPA sólo define la interfaz de la plataforma de agentes, es decir, el comportamiento externo, permitiendo de esta forma que las decisiones de diseño sean tomadas por los equipos de desarrollo. También promueve que los sistemas sean totalmente abiertos, de tal forma, que sistemas heterogéneos puedan interaccionar a nivel de sociedad de agentes. El modelo FIPA establece el modelo lógico referente a la creación, destrucción, registro, localización y comunicación de agentes.

Para ello FIPA define los servicios que deben ser proporcionados por toda plataforma de agentes:

- Un sistema de transporte de mensajes (Internal Platform Message Transport)
- Un sistema de gestión de agentes (Agent Management System - AMS)
- Un servicio de directorio (Directory Facilitador - DF)
- Un canal de comunicaciones para los agentes (Agent Comunication Channel)

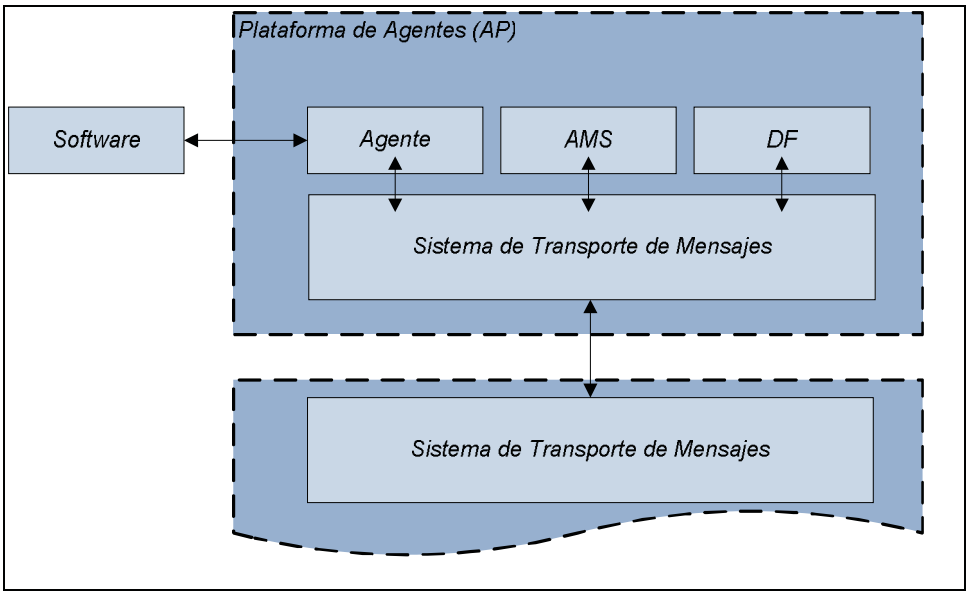

Ilustración 2-1 Modelo de Referencia de Agentes FIPA

El modelo de referencia de agentes FIPA (Ilustración 2 1), proporciona el marco en el cual los agentes operan y existen [González et al., 2006].

- El sistema de gestión de agentes (AMS) proporciona al resto de los agentes el servicio de páginas blancas y además mantiene un directorio que contiene las direcciones de los agentes registrados en la plataforma de agentes (AP).

- El facilitador de directorios (DF) proporciona un servicio de páginas amarillas a aquellos agentes que le consultan buscando los servicios suministrados por otros agentes de la plataforma.

- El servicio de transporte de mensajes (MTS) constituye el método de comunicación por defecto entre los agentes de las diferentes plataformas.

2.4 Infraestructura Agente

La infraestructura agente está destinada a hacer posible la interacción de los agentes, permite a los agentes comunicarse y socializar debido a que entrega las reglas para ello, de tal modo los agentes pueden compartir su

conocimiento y así también entenderse entre ellos. Esta infraestructura comprende tres aspectos principales, las ontologías, los lenguajes de comunicación de agentes y los protocolos de interacción de agentes.

2.4.1 Ontologías

Una ontología corresponde a un modelo que entrega, una comprensión compartida de un dominio de cierto interés, dando un significado formal o semi-formal de los conceptos de un dominio, de sus definiciones e interrelaciones.

Una ontología usualmente contiene:

- Clases, elementos generales en los dominios interesados.
- Propiedades, cualidades de esos elementos (clases).
- Relaciones, lazos entre los diversos elementos (clases).

En cierto aspecto, la ontología asemeja a un modelo Entidad-Relación o un modelo orientado a objetos, pero se diferencia con respecto a ellos, en los siguientes puntos:

- El uso, un modelo E-R (o a objetos) está orientado a la modelación de una base de datos, un proceso o un negocio. Mientras que un importante uso de las ontologías es el de permitir la interoperabilidad entre sistemas distintos.

- El nivel de formalismo, por lo anterior, como modelos capturan características distintas de la misma realidad. En una ontología no se normaliza ni se especifican claves primarias. En vez de ello, se especifican relaciones lógicas (inversoDe, subTipoDe, esSimilarA, etc.) entre los conceptos.

- El lenguaje de modelación, para describir en E-R y a objetos se utilizan diagramas, UML, etc. Para representar ontologías existen distintos lenguajes que se verán a continuación.

Los lenguajes para ontologías nos proveen una forma de describir los conceptos de manera no ambigua, con un nivel apropiado de formalización semántica que otros lenguajes no tienen (como HTML o XML). Los requisitos que deben cumplir estos lenguajes son:

- Permitir compartir ontologías,
- Considerar el ciclo de evolución de las ontologías,

- Apoyar la interoperabilidad entre diversas ontologías,
- Detectar inconsistencias, y
- Proveer expresividad a una variedad amplia de conocimiento.

De esta manera surgen varias iniciativas y a continuación mencionaremos las que han tenido mayor grado de aceptación [Ilustración 2-2]:

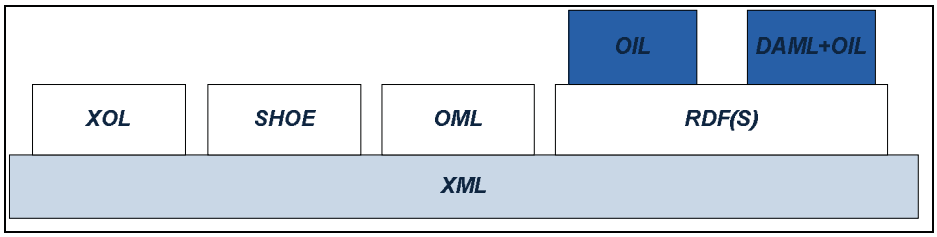

Ilustración 2-2 Lenguajes para ontologías

- XOL: Ontology Exchange Language, comunidad Bioinformática de USA.
- SHOE: Simple HTML, Ontology Extensión, Universidad de Maryland.
- OML: Ontology Markup Language, Universidad de Washington.
- DAML-ONT: DARPA Agent Markup Language, Proyecto DARPA (Defense Advanced Research Project Agency)
- OIL: Ontology Interchange Language, Proyecto de Conocimiento Ontológico (European IST)
- DAML+OIL:W3C
- OWL: Web Ontology Language, W3C.

2.4.2 Comunicación entre agentes

Para coordinar los esfuerzos de cooperación entre los agentes, se vuelve esencial la capacidad de comunicación. La comunicación puede ser analizada desde distintas perspectivas. Para que un acto comunicativo sea llevado a cabo se requiere que se compartan tres aspectos entre los agentes:

- Sintaxis, representan la estructura de los símbolos que son usados para comunicarse, estos son, la secuencia de los símbolos que constituyen el mensaje, relacionados de acuerdo a cierta gramática predefinida.

- Semántica, define el significado de los símbolos para lograr entenderlos.

- Pragmática, representa como los símbolos son interpretados.

Como consecuencia, el significado de una comunicación es en parte el resultado de la semántica, y por otra parte de la pragmática. La ambigüedad está siempre presente en las comunicaciones humano a humano, pero esta debe ser eliminada en una comunicación entre agentes. Dentro de la comunicación entre humanos se distinguen 3 aspectos:

- Locución, corresponde al sonido físico de la persona que habla.

- Ilocución, corresponde al significado de lo que el emisor dijo.

- Perlocución, corresponde al acto que resulta de la locución.

La investigación de comunicaciones no ambiguas en sociedades de agentes ha centrado su atención a modelos basados en las comunicaciones humanas, particularmente a la teoría del acto del habla.

La teoría del acto del habla considera el lenguaje de los humanos como acciones que pueden ser clasificadas en pedidos, sugerencias, obligaciones y respuestas. En esta teoría el término “performative” es usado para capturar y transmitir sin ambigüedad el mensaje. Las palabras que expresan “performatives” son verbos, entre ellos podemos encontrar: decir, solicitar, reportar y demandar.

De este modo, la teoría del acto de hablar, realiza el uso de “performatives”, las cuales apuntan a transmitir al receptor la intención detrás del mensaje, pero esto no garantiza la comprensión del contenido del mensaje, o su correcto procesamiento. La teoría del acto del habla tampoco propone una sintaxis para mensajes, ni un protocolo de comunicación entre los agentes.

Para llevar a cabo la comunicación entre agentes, en el último tiempo han surgido 2 estándares por sobre las demás propuestas, KQML y FIPA ACL.

KQML

KQML (Knowledge Query Manipulation Language) **[Finin et al., 1993]**, es parte del Knowledge Sharing Effort (KSE) y es concebido como un formato para mensajes y un grupo de protocolos. Una de sus principales características es que apoya a los agentes en la identificación y conexión con otros agentes, además del intercambio de información. Además, la sintaxis de KQML se basa en una lista balanceada de paréntesis similar al LISP común. Los principales elemento de un mensaje KQML son:

- Performative: Específica el tipo de acto comunicativo del mensaje.
- Content: Específica el contenido del mensaje.
- Sender: Denota la identidad del agente que envía el mensaje.
- Receiver: Denota la identidad de él(los) agente(s) que deben recibir el mensaje.
- Language: Específica el nombre del lenguaje en el que está escrito el mensaje.
- Ontology: Específica las ontologías utilizadas para dar significado a los símbolos del mensaje.
- 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.

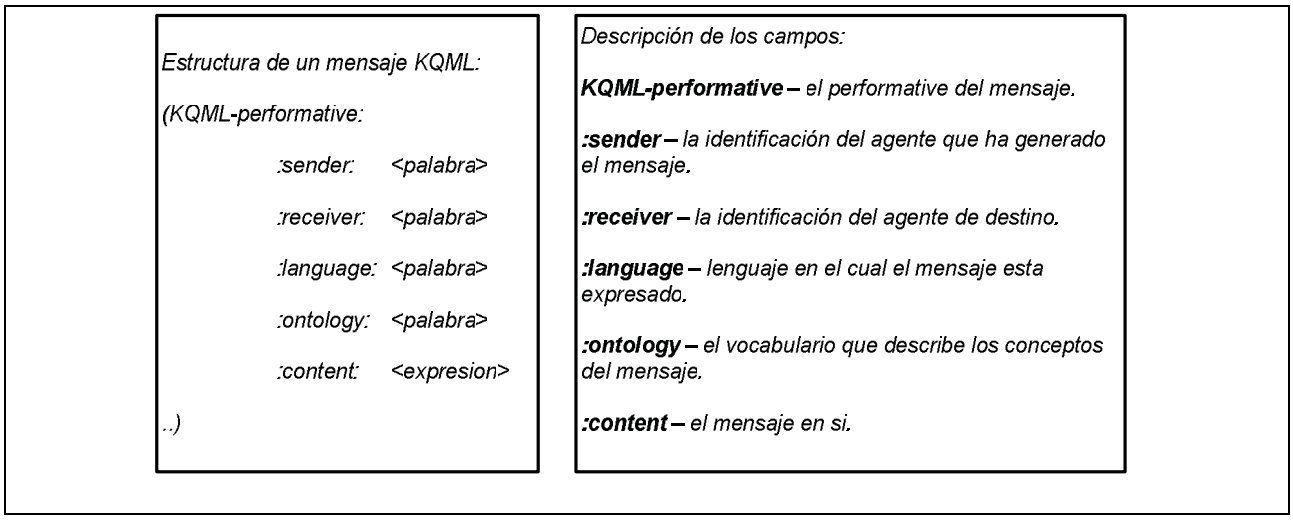

Ilustración 2-3 Mensaje KQML

El protocolo básico está definido por la estructura presentada en la Ilustración 2 3; una completa especificación puede ser encontrada en [Finin et al., 1993].

FIPA ACL

FIPA ACL (Foundation for Intelligent Physical Agents) **[FIPA, 2002]** es un lenguaje de comunicación para agentes que involucra varios entes en la industria y academia. Como estándar, FIPA ACL presenta claramente los elementos prácticos de la comunicación y cooperación inter-agente, además de un fundamento semántico formal bien

definido. A pesar de que los elementos principales son similares a KQML, se diferencian en el grupo de performatives predefinidos. La Ilustración 2-4 presenta un modelo básico de mensaje FIPA ACL.

```
(inform
:sender (agent-identifier :name servBroker)
:receiver (set (agent-identifier :name user03))
:content "
<BrokerAnswerMsg xmlns="http://www.polito.it/Amessage">
<clientRequest ="007736033922"/>
<Service type="LiveCamera" id="33488212157"/>
<location> <uri>"rmi://robo1.polito.it:8480/camera04"</uri></location>
<device> ... </device>
</Service>
</BrokerAnswerMsg>"
:ontology image-device-ontology
:language xml
:in-reply-to r09)
```

Ilustración 2-4 Mensaje FIPA ACL

Sin embargo últimamente se han encontrado una serie de limitantes:

- No entrega ayuda para requerimientos de tiempo real y rendimiento, útil en aplicaciones de telecomunicaciones. Los mensajes debieran incluir propiedades temporales o de razonamiento temporal.

- No entrega soporte para tolerancia a fallas o calidad de servicio que podrían ser de interés en sistemas en tiempo real.

- Cuando un agente envía un mensaje a otro, este estándar FIPA no garantiza la real realización del resultado previsto por el mensaje (efecto racional).

2.4.3 Protocolos de interacción de agentes (AIPs)

La estructura de comunicación de los agentes es similar a secuencias de mensajes, estas secuencias son denominadas protocolos de comunicación o de interacción. Una definición más formal de este concepto a continuación:

"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 et al., 2001]

Los protocolos de interacción son usados por las comunidades de agentes cooperantes para determinar y mantener metas compartidas, para determinar tareas comunes, para resolver conflictos, y para transmitir su conocimiento. Básicamente, los protocolos de interacción se necesitan para mantener una consistencia global sin comprometer la autonomía de los agentes. Los protocolos de interacción definen la conversación entre 2 o más agentes, que es, la secuencia de posibles mensajes para intercambiar desde un dominio preestablecido en el dominio de un ACL. Una sociedad de agentes puede compartir un gran conjunto de protocolos de interacción, dependiendo de la tipología de las relaciones que la comunidad espera manejar.

FIPA ha definido una serie de protocolos, pero cada desarrollador esta libre para crear los protocolos que estime conveniente. Para definir estos protocolos FIPA definió AUML, que es un lenguaje para representar protocolos y que usa como base el lenguaje UML. Dentro de AUML se pueden distinguir los siguientes elementos dentro de los diagramas:

- Roles de Agentes: Indica el rol de cada uno de los agentes dentro de la comunicación.
- Línea de Vida: Define cuanto tiempo participa el agente dentro de la comunicación.
- Hilos de interacción: Muestra el instante de tiempo en que están realizando una tarea.
- Mensajes: Es la representación de una acción.

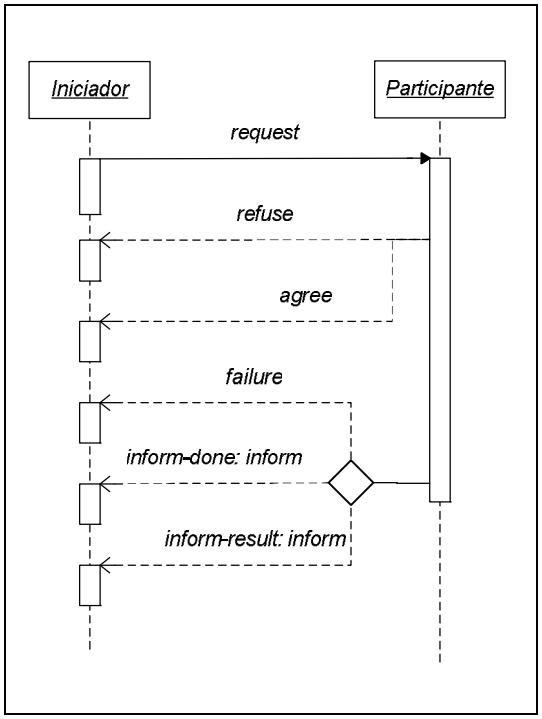

Ilustración 2-5 Protocolo de interacción Request de FIPA

Un ejemplo de protocolo de interacción es el Protocolo de Interacción Request FIPA [FIPA, 2002], el cual es presentado en la Ilustración 2 5, el cual permite a un agente (el iniciador) pedir a otro (el participante) realizar alguna acción. El participante después de solicitar el pedido, decide si aceptar o rechazar el pedido, y respectivamente la acción condición de aceptado o denegado se vuelve verdad. Si la condición de rechazado es verdadera, un mensaje de rechazado se retorna al iniciador, de otro modo, si la condición de aceptado es verdadera, entonces un mensaje de aceptado es enviado. Después de realizado el pedido (en el caso de aceptarlo) el mensaje con el resultado del pedido es enviada al Iniciador, o se envía un mensaje de error si la operación falla.

FIPA ha estandarizado una serie de protocolos los cuales son dados a conocer a continuación:

- Subscribe: El agente pide ser notificado de una condición y que se le mantenga informado de los cambios que tenga.

- Request: Solicitar a un agente que realice una tarea.

- Request When: Solicitar a un agente que haga una tarea cuando se cumpla alguna condición.

- Contract Net: Un agente manager pide un presupuesto a un conjunto de agentes para desarrollar una tarea, cada uno de ellos le entrega su presupuesto y el manager procede a elegir al agente que realizará la tarea.

- Brokering: Un agente ofrece servicios de otros agentes, o reenvía las peticiones al agente adecuado.

- Subasta Inglesa: Los agentes participan en una subasta que se inicia en un precio bajo y va subiendo progresivamente, no es necesario que el precio suba en cada ronda como en la subasta inglesa real.

- Subasta holandesa: Los agentes participan en una subasta que se inicia en un precio alto y va bajando progresivamente, no es necesario que el precio baje en cada ronda como en la subasta holandesa real.

- Recruiting: Trabaja de la misma forma que el brokering pero las respuestas son directas al agente que las necesita.

- Propose: Un iniciador propone un grupo de agentes para realizar una tarea y estos aceptan o no.

2.5 Movilidad de los agentes

El concepto de movilidad no es nuevo en la ingeniería de software, y no se restringe solamente al mundo de los agentes. Los agentes móviles son agentes que pueden moverse entre entornos. Sin embargo, la autonomía que los caracteriza hace la diferencia entre los agentes móviles y otros tipos de códigos móviles. De hecho, los agentes pueden decidir cuándo y dónde moverse, exhibiendo de este modo la tan llamada capacidad de moverse proactivamente.

Dos tipos de movilidad son identificados [Cubillos, 2002], los cuales caracterizan a todos los tipos de códigos móviles, incluidos los sistemas de agentes:

- Movilidad fuerte, es la capacidad de un sistema de código móvil moverse llevando consigo el código y el estado de ejecución actual al destino, y allí continuar su ejecución.

- Movilidad débil, es la habilidad de transferir el código a través de distintos entornos. En este caso el código puede ser acompañado de datos de iniciación, pero el estado de la ejecución no es transferido.

Los sistemas basados en agentes móviles, pueden ser útiles en muchas áreas de aplicación. Por ejemplo, pueden ser utilizados para recuperar información, donde los agentes son diseñados para visitar varios nodos de una red, con la orden de recuperar algún tipo de información. El camino que los nodos visitan es llamado itinerario, el cual puede ser fijado al inicio del recorrido, o ser definido en forma autónoma por el agente durante su viaje. La información recuperada por los agentes permite reducir el flujo de datos dentro de la red, debido a que los agentes pueden buscar localmente toda la información ofrecida por el host.

Otra aplicación propuesta para los agentes móviles está en el campo del comercio electrónico, en el cual los agentes viajan entre diferentes hosts llamados mercados virtuales, con el objetivo de intercambiar bienes o servicios, en nombre del usuario. Los agentes en el comercio electrónico, pueden aplicar su inteligencia no sólo para alcanzar tratos, sino que también, pueden ayudar a evaluar bienes o servicios, o comparar precios.

Los agentes móviles también son vistos como una tecnología prometedora para la administración de redes. La idea es que los agentes estén preparados para recuperar y analizar información de cada nodo en la red, y entonces puedan redefinir la configuración local del nodo, o puedan proveer actualización del software en el nodo. De este modo, la administración de la red es más escalable y confiable.

Otro uso considera a los agentes móviles para balancear la carga en sistemas distribuidos. Bajo este paradigma, los componentes del sistema son delegados a agentes autónomos, que tienen la capacidad de interrumpir el proceso y viajar a un entorno más conveniente en la red. Esta aplicación requiere la implementación de una movilidad fuerte de código, debido a que el proceso debe ser restaurado después del viaje, en el punto exacto en el cual fue interrumpido.

Tomando en consideración los ejemplos mostrados anteriormente, se puede observar que la movilidad ofrece una característica muy útil, que puede ser explotada por la ingeniería de software en la creación de agentes basados en sistemas distribuidos. Sin embargo, debe incurrirse en un gasto adicional asociado a la coordinación de agentes móviles, y el uso de recursos. Además se abre un gran y complejo problema de seguridad.

2.6 Plataforma de desarrollo

En los últimos años se han desarrollado diversas plataformas de agentes y toolkits relacionados con las tecnologías orientadas a agentes para su uso en investigación, educación y fines industriales. Una completa colección con más de cien productos ofrecido puede verse en [AgentLink, 2009]. En esta sección se presentaran las plataformas disponibles, agrupándolas de acuerdo a sus características en común, como pueden ser la funcionalidad

objetivo que fue concebida que debían soportar, o las capacidades específicas que ofrecen. Sin embargo, es necesario aclarar que la siguiente categorización no está completa ni es definitiva.

2.6.1 Plataformas centradas en el usuario

Algunas plataformas de agentes son concebidas para soportar directamente la interacción del usuario con su entorno. Por ejemplo, Dejima [Hodjat y Amamiva, 2002] concentra su esfuerzo en permitir a los usuarios comandar y controlar una o más aplicaciones, ocultando la complejidad de la aplicación y haciendo al usuario el eje central.

2.6.2 Plataformas orientadas a la simulación

Un importante grupo de plataformas para agentes fue concebido para la construcción de sistemas de simulación. Estos utilizan herramientas que proveen a los agentes de un alto nivel de programación, ofreciendo al usuario interfaces visuales que permiten la fácil construcción de sistemas interactivos. Estos entornos son propuestos para construir software de entrenamiento, para desarrollar herramientas educacionales, para construir juegos, y para probar algoritmos de IA, entre otros. El modelo de programación no es único, por eso ellos proveen su propio lenguaje y arquitectura de agentes, pero en general ellos buscan proveer los medios para la creación de agentes que puedan adaptarse rápidamente a aplicaciones impredecibles. AgenSheets, EXCALIBUR, y Direct IA son tres ejemplos de este tipo de plataformas.

Un caso especial de plataforma orientada a la simulación es Evo [Krumpus, 2001], el cual fue diseñado para implementar una simulación compleja de la vida, usando operadores biológicos para recombinar y mutar agentes, que actúan de una forma similar a los algoritmos genéticos. En Evo un genoma de un agente individual es el programa que es ejecutado, y la aplicación de los operadores causa la evolución en sus comportamientos.

2.6.3 Plataformas centradas en la inteligencia

Este tipo de plataformas provee los medios para la implementación de los agentes que puedan aportar una (o más) arquitecturas concretas. Estos sistemas también ofrecen soporte explicito para la distribución de la inteligencia en la construcción de aplicaciones multiagente. Un ejemplo en esta categoría es JACK, un framework para construir agentes usando la arquitectura BDI. Este soporta la colaboración de los agentes a través de la especificación de un comportamiento global que es traducido automáticamente en actividades asignadas a los diferentes roles de los agentes participantes.

dMARS, un precursor de JACK, es otro ejemplo de la implementación de la arquitectura BDI, la cual sin embargo, está enfocada en un agente individual. Un ejemplo adicional es iGEN agentes cognitivos embebidos.

2.6.4 Plataformas centradas en la movilidad

La movilidad fue considerada la propiedad central en varias plataformas de agentes. Estas plataformas pueden implementar movilidad fuerte o débil, usualmente con el soporte de un lenguaje interpretado.

Dos plataformas que ofrecen fuerte movilidad de los agentes son Telescript y ARA. En Telescript los agentes son situados en lugares, y los lugares y agentes son programados en un lenguaje intermedio portable llamado Low Telescript, el cual es ejecutado por los motores. Por otro lado, ARA, administra los agentes por medio de un sistema central de lenguaje independiente, y lo interpreta por medio de algunos lenguajes de programación tradicionales (como C/C++ y Tcl).

Otro ejemplo de plataformas móviles es TACOMA, el cual permite la creación de agentes que implementan la movilidad débil, usando una extensión del lenguaje Tcl. Agent Tcl, es una plataforma que soporta la movilidad fuerte, y que también fue desarrollado usando el mismo lenguaje. Ambos, TACOMA y Agent Tcl implementan los agentes como procesos Unix en el interpretador Tcl. Como último ejemplo en esta categoría, mencionaremos a Aglets un framework que implementa a los agentes como hilos en el lenguaje Java y ofrece movilidad débil.

2.6.5 Plataformas de propósito general

Otro grupo de plataformas ofrecen las características de todas las anteriores, y pueden ser consideradas como plataformas de propósito general. La mayoría de estas plataformas adoptan los estándares propuestos por FIPA.

JADE [Bellifemine et al., 1999] es un ejemplo representativo de este grupo, y una de las plataformas mas referidas en los libros. JADE es implementado en el lenguaje de programación JAVA, el cual provee el soporte para la movilidad de los agentes. Los agentes pueden ser programados adoptando las arquitecturas reactivas y BDI, y soporta la integración con recursos de software externo en las tareas del agente. JADE provee un entorno de ejecución para los agentes llamada contenedor. Una extensión llamada JADE-LEAP permite la ejecución de entornos en pequeños dispositivos, como las PDAs.

JADE facilita la implementación de los sistemas multiagente gracias a una capa intermedia la cual cumple con la especificación FIPA, y además provee un conjunto de herramientas gráficas, mostrado en la ilustración, que soportan las fases de desarrollo y depuración. La plataforma puede ser distribuida a través de varias máquinas, independiente del sistema operativo que posea gracias a que está desarrollado en Java.

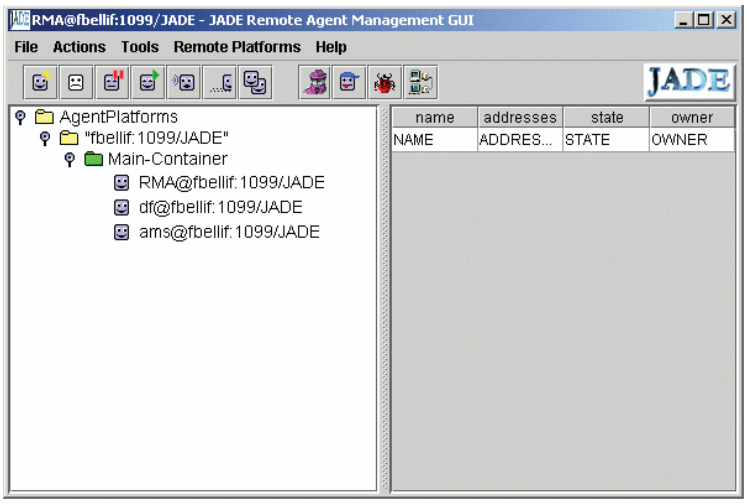

Ilustración 2-6 Interfaz gráfica de control proporcionada por JADE

Otro ejemplo de plataforma para propósito general es GPP, en la cual se trabajará en este proyecto, y que será detallada en otro capítulo, dentro del marco provisto por GPP , este se encarga de controlar el ciclo de vida de los agentes, así como también prestar los servicios necesarios para su funcionamiento (movilidad, comunicación, etc.). A diferencia de JADE no se rige por una arquitectura de agentes fija, es por ello que puede adoptar cualquier estándar inclusive el mismo FIPA.

2.7 Metodologías para el desarrollo de sistemas multiagente

Diversas metodologías se han alzado para llevar a cabo el proceso de desarrollo de un MAS. Estas surgen como una forma de abarcar los ámbitos que las metodologías actuales no cubren en el desarrollo de un MAS como son las nociones de autonomía e inteligencia. Por ello nos encontramos con un gran variedad, por nombrar sólo algunas, AOR, Cassiopeia, BDI, AUML, PASSI, Message, Vowel Engineering, Gaia, MAS-CommonKADS, MaSE, MASSIVE, INGENIAS, Tropos, RoMAS, entre otras.

Sin embargo, algunas de las nombradas anteriormente han sido aceptadas en mayor grado que otras por lo cual a continuación se describirán algunas de las más populares.

2.7.1 MaSE

MaSE (Multi agent systems Software Engineering) adopta el paradigma orientado a objetos y considera al agente como sólo una especialización de un objeto. La especialización que considera MaSE parte de la premisa que los agentes se coordinan unos con otros vía conversaciones y actúan de manera proactiva para alcanzar metas individuales y globales.

En MaSE los agentes son procesos de software que interactúan entre sí con el fin de alcanzar una meta global. Son sólo una abstracción conveniente que puede o no poseer inteligencia. En este sentido, los componentes inteligentes y no inteligentes se gestionan igualmente dentro del mismo marco. El proceso de desarrollo en MaSE es un conjunto de pasos, la mayoría de los cuales pueden realizarse utilizando la herramienta AgentTool, debido a que es soportada por MaSE.

El análisis en MaSE consta de tres pasos: capturar las metas, capturar los casos de uso y definir los roles. Como productos de esta etapa se esperan: diagramas de objetivos, que representan los requisitos funcionales del sistema; diagramas de roles, que identifican los roles de los agentes, la tareas asociadas a los roles y las comunicaciones entre roles y entre tareas; y casos de uso, no mostrados como diagramas sino como una enumeración considerando la posibilidad de usar diagramas de secuencia para detallarlos.

El diseño consta de cuatro pasos: crear clases de agentes, construir conversaciones, ensamblar clases de agentes y diseño del sistema. Como productos de estas etapas, se esperan: diagramas de clases de agentes, que enumeran los agentes del sistema, roles jugados e identifican conversaciones entre los mismos; descomposición del agente en los componentes y definir la arquitectura del agente mediante los componentes; diagramas UML de despliegue para indicar cuantos agentes habría en el sistema y de qué tipo.

2.7.2 GAIA

Propone trabajar inicialmente con un análisis de alto nivel. En este análisis se usan 2 modelos, el modelo de roles que permite identificar los roles clave en el sistema junto a sus propiedades definitorias y el modelo de interacciones que define las interacciones mediante una referencia a un modelo institucionalizado de intercambio de mensajes, como los protocolos FIPA. Inmediatamente después de esta etapa, se procede a lo que GAIA considera diseño de alto nivel. El objetivo de este diseño es generar 3 modelos: el modelo de agentes que define los tipos de agentes que existen, cuantas instancias de cada tipo y que papeles juega cada agente, el modelo de servicios que

identifica los servicios (funciones del agente) asociados a cada rol, y un modelo de conocidos, que define los enlaces de comunicaciones que existen entre los agentes.

A partir de esto, los autores de GAIA proponen aplicar técnicas clásicas de diseño orientado a objetos. Sin embargo, GAIA declara que queda fuera de su ámbito. Esto debido a que esta metodología busca especificar como una sociedad de agentes colabora para alcanzar los objetivos del sistema, y que se requiere de cada uno para lograrlo.

2.7.3 PASSI

PASSI (Process for Agent Societies Specification and Implementation) **[Cossentino y Potts]** es una metodología paso a paso para elaborar un proyecto orientado a agentes. Está destinada a diseñar y desarrollar sociedades multiagente integrando modelos de diseño y conceptos desde la ingeniería de software orientada a objetos y los sistemas multiagente usando notación UML. Los modelos y fases de PASSI abarcan la representación antropomórfica de los requerimientos del sistema, del punto de vista social, de la solución arquitectónica, de la producción de código y reutilización, y de la configuración de despliegue que soporta la movilidad de los agentes.

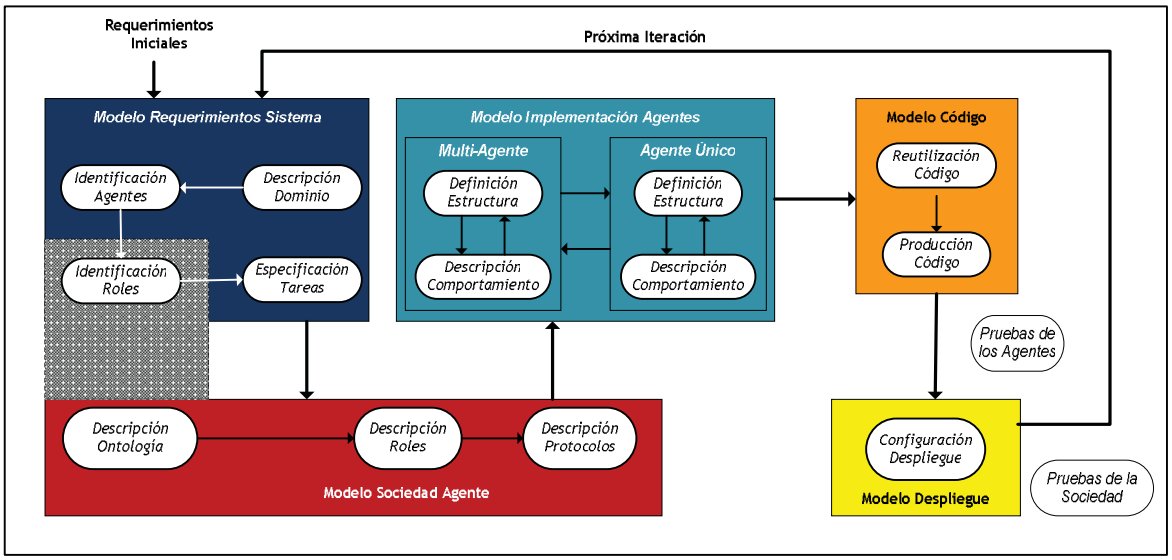

Ilustración 2-7 Diagrama de Modelos y Fases de PASSI

La metodología PASSI se compone de cinco modelos referentes a diversos niveles de diseño, y de doce pasos en el proceso de construcción de un sistema multiagente. En la Ilustración 2-7 Diagrama de Modelos y Fases

de PASSI se muestra una descripción gráfica de PASSI. A continuación se realiza una breve descripción de la metodología.

Modelo de Requerimientos del Sistema, corresponde a un modelo antropomórfico de los requerimientos del sistema en términos de agentes y objetivos. Este modelo se compone de las siguientes fases:

1. Fase de Descripción del Dominio, una descripción funcional del sistema usando casos de uso convencionales.

2. Fase de Identificación de Agentes, Separación de la responsabilidad que concierne a los agentes representados como paquetes UML.

3. Fase de Identificación de Roles, Uso de diagramas de secuencia para explorar la responsabilidad de cada agente a través de escenarios.

4. Fase de Especificación de Tareas, Especificación de las capacidades de cada agente a través de diagramas de actividad.

Modelo de Sociedad de Agentes, es un modelo de las interacciones y dependencias sociales entre los agentes que participan en la solución. Está compuesto por cuatro fases:

5. Fase de Descripción de la Ontología, uso de diagramas de clases y restricciones OCL para describir el conocimiento individual de los agentes y la pragmática de sus acciones.

6. Fase de Descripción de Roles, Uso de diagramas de clases para mostrar los distintos roles de cada agente, las áreas que involucran a esos roles, las capacidades de comunicación y las dependencias entre agentes.

Fase de Identificación de Roles de Tareas, Ver el Modelo de Requerimientos del Sistema.

Fase de Descripción de Protocolos, Uso de diagramas de secuencia para especificar la gramática de cada protocolo de comunicación en términos de los performatives.

Modelo de Implementación de Agentes: Se trata de un modelo de la solución arquitectónica en términos de clase y métodos. Compuesto por dos fases:

7. Fase de Definición de la Estructura de Agentes, uso de diagramas de clases convencionales para describir la estructura de solución.
8. Fase de Descripción del Comportamiento de los Agentes, uso de diagramas de actividad o diagramas de estado para describir los comportamientos individuales de los agentes.

Modelo de Código: Este es un modelo de la solución a nivel de código. Compuesto por dos fases:

9. Fase de Librería de Reutilización de Código, una librería de clases y diagramas de actividad con código reutilizable asociado.

10. Fase de Descripción del Comportamiento de los Agentes, código fuente del sistema objetivo.

Modelo de Despliegue: Es un modelo de la distribución de las partes del sistema a través de las unidades de proceso del hardware y de su migración. Compuesto solamente por:

Fase de Configuración de Despliegue, uso de diagramas de despliegue para describir la posición de los agentes para permitir el procesamiento de unidades y cualquier restricción con respecto a la migración y movilidad.

- Testeo, la actividad de testeo se ha dividido en 2 pasos diferentes: el testeo individual de los agentes el cual comprueba que los comportamientos realicen las actividades para las cuales fueron diseñados. Y el testeo de la sociedad, el cual valida el correcto funcionamiento de las interacciones que realizan los agentes, para de esta manera verificar que cooperen en la solución de problemas.

3 MOTORES GRÁFICOS 3D

El Motor gráfico es la parte principal de un programa 3D, es el encargado de gestionar y actualizar los gráficos 3D (renderizar) en tiempo real, con esto se logra la sensación de movimiento. Entre los motores gráficos más utilizados se encuentran los de Quake y de Unreal.

La utilización de estos no se limita a la creación de videojuegos, sino que puede ser aplicado a cualquier tipo de aplicación que requiera mostrar algo más que una imagen 2D como puede ser algún simulador de actividades, etc. El rendimiento total del software a desarrollar esta fuertemente relacionado con el motor gráfico a utilizar.

El funcionamiento de un motor grafico a grandes rasgos es enviar rápidamente al CPU los datos necesarios para calcular el próximo frame a ser presentado en pantalla. En caso de utilización de física en la aplicación o control de colisiones, el mismo motor será el encargado de realizar los cálculos para determinar el peso, o posición actual del objeto 3D.

Los motores gráficos facilitan la programación de juegos y aplicaciones, ya que ocultan todo el proceso de cálculo y despliegue de imágenes, permitiendo a los programadores centrarse más en las partes importantes del software como es la lógica, más que en los problemas de bajo nivel respecto a gráficas.

3.1 Técnicas utilizadas

A continuación se muestran algunas técnicas utilizadas en los motores gráficos para la muestra de imágenes. **[Zerbst y Düvel,2004]**

- **Renderizado**

  Proceso de conversión de dibujo 3D con texturas y luz hecho por el computador. El proceso entiende las gráficas de los juegos de 3D como coordenadas en un plano cartesiano de tres dimensiones.

- **Arboles BSP**

  Es una estructura de datos usados para organizar objetos dentro de un espacio. Tiene aplicaciones en la remoción de áreas ocultas y en el trazado de rayos

- **Radiosidad**

Técnica para el cálculo de la iluminación global de un ambiente cerrado. La idea en que se basa esta técnica es buscar el equilibrio de la energía que es emitida por los objetos emisores de luz y la energía que es absorbida por los objetos en el ambiente.

- **MipMapping**

Técnica de manejo de texturas que cambia la textura de un polígono en un objeto 3D dentro de un juego según el ángulo de vista del jugador ó las condiciones del juego. Conforme nos acercamos a un objeto, éste gana calidad. De esta forma objetos alejados tendrán un nivel de resolución bajo y objetos cercanos alto. Conseguimos mayor rendimiento.

- **Phong y Gourand**

Gourand se basa en que los polígonos aproximan una superficie curva. Se calculan las intensidades de los vértices. Pong interpola las normales en lugar de las intensidades

- **Bump-Mapping**

Se utiliza para agregar el detalle a una imagen sin aumentar el número de polígonos. Nos basamos en algoritmos que captan la textura inicial de la imagen y la convierten en otra. Crea pequeños Bump mapping en la superficie del objeto para darle texturas sin cambiar la superficie del objeto.

- **Lightmas**

Esta técnica se empezó a usar en el 1996, y la crearon la gente de Id Software en el Quake. Los lightmaps (mapas de luz) simplemente consisten en añadir una segunda textura a todas y cada una de las caras existentes en una escena 3D. Es un buen método para ahorrarnos el uso de la Radiosidad.

3.2 Listado Motores Gráficos

A continuación se describirán brevemente una serie de distintos motores gráficos:

- **Unreal Engine**

Unreal Engine **[Unreal]** es un motor para juegos de PC y consolas creados por la compañía Epic Games. Implementado inicialmente en el shooter en primera persona llamado Unreal en 1998, siendo la base de muchos juegos desde entonces. También se ha utilizado en otros géneros como el rol y juegos de perspectiva en tercera persona. Está escrito en C++, creando varias versiones que engloban las plataformas PC (Microsoft Windows, GNU/Linux), Apple Macintosh (Mac Os, Mac Os X) y la mayoría de consolas

(Dreamcast, Xbox, Xbox 360, Playstation 2, Playstation 3, Wii). Unreal Engine también ofrece varias herramientas adicionales de gran ayuda para diseñadores y artistas. Está basado en una extensión del renderizado en portales conocido como Dynamic Scene Graph Technology (DSG), BSP y radiosidad. Su principal característica es la gran escalabilidad.

- **Crystal Space**

Crystal Space [Crystal Space] fue desarrollado por Jorrit Tyberghein, basado en renderizado en portales, BSP, ZBuffering y radiosidad. Entre todas sus principales características destacamos la gran portabilidad y la gran escalabilidad proporcionada por el sistema de plugins y, además, bajo licencia LGPL. Las plataformas soportadas son UNIX (Linux, y Solaris), DOS, Macintosh, Amiga, Windows, BeOS, NextStep, Rhapsody y ports OpenStep.

- **Fly3D**

Desarrollado por Paralelo Computaçao, basado en renderizado por árboles BSP, PVS y portales, entre sus características destacamos la escalabilidad proporcionada por su sistema de plugins. La plataforma que soporta es Windows.

- **Genesis3D**

Desarrollado por Eclipse Entertainment basado en renderizado en portales, BSP y radiosidad. La única plataforma soportada es Windows.

- **Torque (v12)**

Motor gráfico utilizado en el juego Tribes 2 de Dinamix basado en renderizado en portales.

- **Quake 2**

Es el motor gráfico del videojuego Quake 2, desarrollado por John Carmak de Id.Software. Se basa en el renderizado por árboles BSP y radiosidad.

- **Java3D**

Aunque Java3D **[Java3d]** no es exactamente un motor gráfico como tal, si cabe mencionarlo ya es que la principal librería proporcionada oficialmente por Sun para desarrollo de aplicaciones 3D. Es un set de bibliotecas que ofrece muchas características para el desarrollo de programas en Java que incorporan gráficas 3D. Debido a haber sido desarrollado por los creadores originales de Java cuenta con la confianza de varios programadores. Java 3D comparte muchas características con bibliotecas gráficas existentes, sin embargo algunas han sido desarrolladas por incipientes procedimientos.

Lejos la mayor ventaja que posee Java 3D para los desarrolladores Java, es que les permite programar cien por ciento en Java. Es por lo tanto muy atractivo tener todo el código de la aplicación, persistencia, y código de la interfaz de usuario (UI) en un lenguaje portable, tal como lo es Java.

A pesar de la promesa que mantiene Sun de escribir una vez y ejecutar en cualquier lado se puede entender más como un sueño de marketing que algo real, especialmente para programas que residen del lado del cliente, Java ha logrado irrumpir de forma importante debido que sus aplicaciones pueden ser ejecutadas fácilmente entre plataformas. Las plataformas de mayor interés hoy en día son Microsoft Windows, Sun Solaris, LINUX, and Macintosh OS X.

- **JMonkeyEngine (JME)**

Java Monkey Engine (JME) [JMonkey] es un framework desarrollado en java, basado en multiples APIs gráficas. Usa una capa de abstracción, lo que permite adherirle cualquier sistema de renderización. Actualmente utiliza LWJGL y JOGL. Es totalmente open-source bajo licencia BSD, por lo que puede ser utilizado en aplicaciones gratuitas y comerciales.

JME es una arquitectura basada en escenografías. La escenografía permite la organización de los datos del programa en una estructura de árbol, donde un nodo padre puede contener cualquier cantidad de nodos hijos, pero un hijo solo un padre. Normalmente estos nodos se organizan espacialmente, lo que permite la rápida eliminación de una rama que represente un conjunto de modelos relacionados.

4 MODELOS 3D

Un modelo 3D puede "verse" de dos formas distintas. Desde un punto de vista técnico, es un grupo de fórmulas matemáticas que describen un "mundo" en tres dimensiones.

Desde un punto de vista visual, un modelo en 3D es un representación esquemática visible a través de un conjunto de objetos, elementos y propiedades que, una vez procesados (renderización), se convertirán en una imagen en 3D o una animación 3D.

Con el fin de crear estos modelos se cuenta con varias técnicas que consisten en la utilización de diferentes modelos de estructuras básicas divididas en 3 tipos [Modelado3d]:

- **Primitivas**: Caja, Cono, Esfera, Geo Esfera, Cilindro, Tubo, Anillo, Pirámide, Tetera, y Plano.

- **Primitivas extendidas:** Hedra, Nudo Toroide, Caja "redondeada", Cilindro "redondeado", Tanque de Aceite, Capsula, Sprindle, Forma L, Gengon, Forma C, Anillo ondulado, Hose, Prisma.

- **Librerías:** Son formas armadas, disponibles según la herramienta a utilizar, estas pueden ser; Puertas, Ventanas, Árboles, Escaleras, etc.

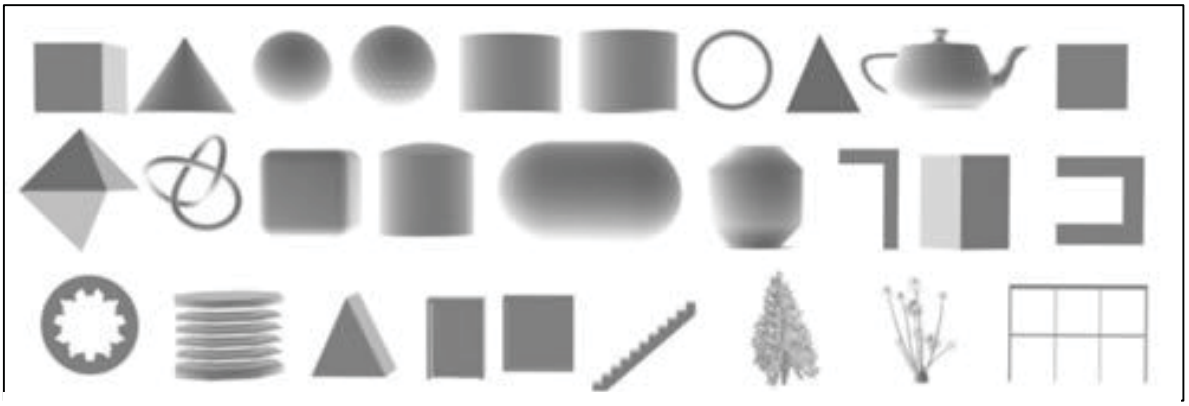

Ilustración 4-1 Figuras Básicas

Todas estas estructuras sirven para poder modelar objetos o escenas más complejas a partir de ellas.

4.1 Técnicas de Modelado

Para lograr el modelado de formas más complejas, es necesaria la mezcla de las figuras básicas, para esto existen diferentes técnicas, algunas de las cuales se nombrarán a continuación:

- **Box Modeling**

Como su nombre lo indica, es el modelado de figuras complejas a partir de un cubo, utilizando algún programa para modificar la malla del cubo hasta lograr la forma deseada.

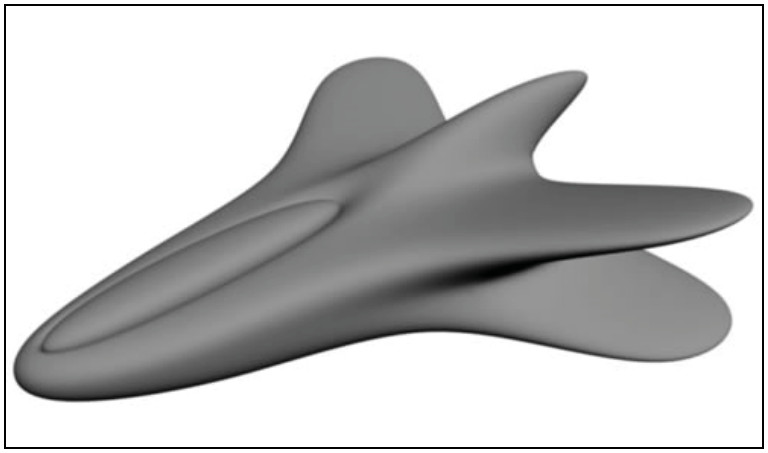

Ilustración 4-2 Ejemplo Box Modeling

- **NURBS Modeling**

Es una técnica para construir mallas de alta complejidad, de aspecto orgánico ó curvado, que emplea como punto de partida splines (figuras 2d) para mediante diversos métodos, crear la malla 3d anidando los splines.

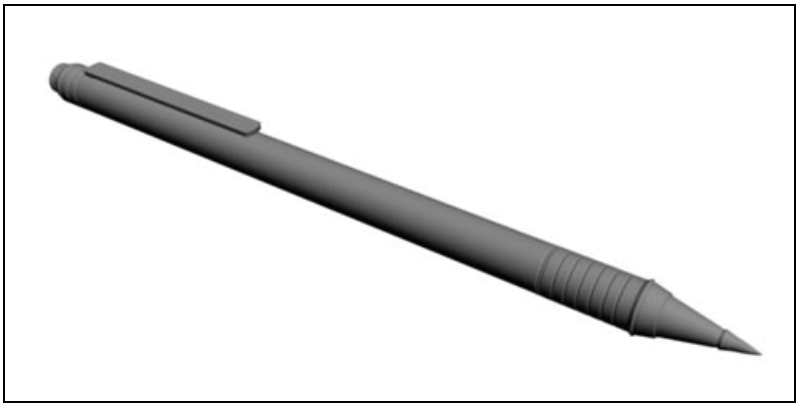

Ilustración 4-3 Ejemplo NURBS Modeling

- **Operaciones Booleanas**

  Consiste, en tomar dos mallas y aplicarles una de tres siguientes operaciones booleanas:

  a) **Resta**: resta dos figuras A – B ó B – A.
  b) **Intersección**: da como resultado sólo lo que esta "tocándose" de ambas figuras.
  c) **Unión**: ambas figuras creando una única nueva.

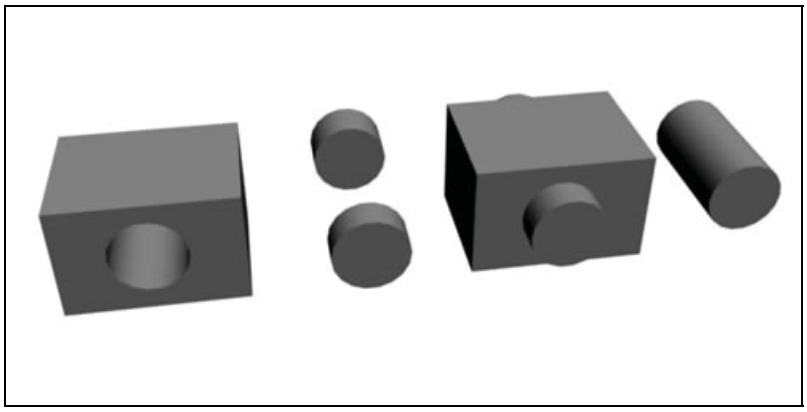

Ilustración 4-4 Ejemplo Operaciones Booleanas

- **Extrude || Lathe**

  Son dos técnicas que a partir, de una figura 2d (spline) crea el volumen.

  - **Extrude**: Da profundidad a un objeto 2d. Extiende la profundidad.
  - **Lathe**: Tomando un spline, lo reproduce por un eje en toda su rotación. Ideal para botellas, copas, y demás objetos sin diferencia en sus costados. Aunque puede combinarse con otra técnica luego, y crear por ejemplo, una tasa.

Ilustración 4-5 Ejemplo Extrude

Ilustración 4-6 Ejemplo Lathe

- **Loft**

Se deben emplear 2 ó más splines, para crear una malla 3d continua. El primer spline, funciona como path (camino) mientras que los demás, dan forma, extendiéndose, a traves del path. Ideal para crear cables, botellas, etc.

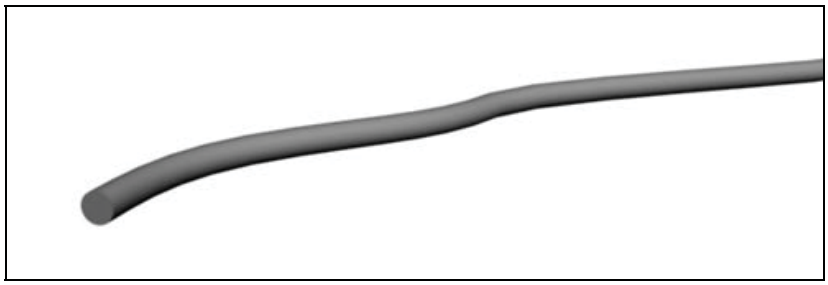

Ilustración 4-7 Ejemplo Loft

- **Sistemas de Particulas**

Es como su nombre lo indica, un sistema de partículas (proyección de formas geométricas, de forma controlada mediante parámetros varios tales como choque, fricción y demás). Es combinable, con efectos de dinámica y deformadores. Es ideal para crear humo, agua, ó cualquier cosa que sea muchos objetos y repetitivos.

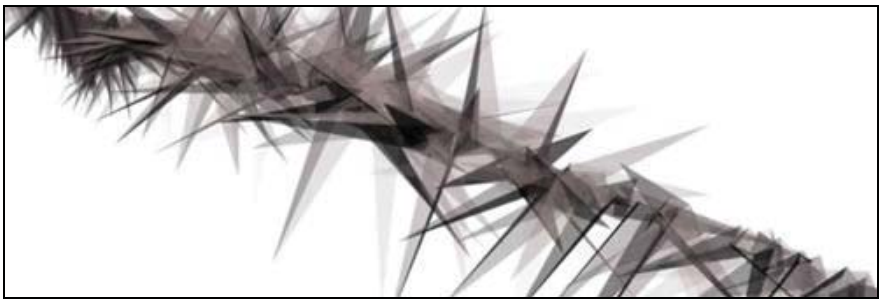

Ilustración 4-8 Ejemplo Sistema de Particulas

- **Modelo por Texturas**

Este tipo de modelado, si es que se lo puede denominar así, en vez de emplear deformadores en la malla, engañan la vista, con mapas del canal alpha (transparencia) para crear recortes, ó engaños

directos de relieve (con un canal especial para esto independiente del de relieve) para crear terrenos por ejemplo.

Ilustración 4-9 Ejemplo Modelo por Texturas

4.2 Herramientas de Modelado 3D

Los motores gráficos permiten la carga de modelos 3d directamente desde un archivo creado por un programa externo. Esto da las facilidades de modelarlo en alguna aplicación con interfaz grafica y herramientas específicas para este fin, entregando mayor calidad al producto final. Entre las diferentes alternativas existentes se destacan las presentadas a continuación.

- **3dStudioMax**

  Es un programa [3dStudioMax] de creación de gráficos y animación 3D desarrollado por Autodesk Media & Entertainment. Fue desarrollado como sucesor para sistemas operativos Win32 del 3D Studio creado para DOS. Kinetix fue más tarde fusionada con la última adquisición de Autodesk, Discreet Logic. 3d Studio Max es uno de los programas de animación 3D más utilizados. Dispone de una sólida capacidad de edición, una omnipresente arquitectura de plugins y una larga tradición en plataformas Microsoft Windows. 3ds Max es utilizado en mayor medida por los desarrolladores de videojuegos, aunque también en el desarrollo de proyectos de animación como películas o anuncios de televisión, efectos especiales y en arquitectura.

- **Sketchup**

Es un programa computacional **[Sketchup]** de modelaje y diseño en 3D (tercera dimensión) encaminado a la arquitectura, desarrollo de videojuegos, películas, ingeniería civil o simple entretenimiento personal. También se ha comenzado a trabajar con pre-adolescentes para despertar en ellos el interés de crear y usar esta herramienta. Los edificios creados con el programa pueden ser geo-referenciados y colocados sobre las imágenes de Google Earth. Los modelos pueden ser subidos a la red mediante el propio programa Google SketchUp y directamente almacenarse en la base de datos 3d Warehouse para ser compartidos.

- **Blender**

Es un programa multiplataforma [Blender], dedicado especialmente al modelado, animación y creación de gráficos tridimensionales. El programa fue inicialmente distribuido de forma gratuita pero sin el código fuente, con un manual disponible para la venta, aunque posteriormente pasó a ser software libre. Actualmente es compatible con todas las versiones de Windows, Mac OS X, Linux, Solaris, FreeBSD e IRIX.

5 SELECCIÓN DE SISTEMA MULTIAGENTE

Para la integración de Graficas 3D a un Sistema Multiagente, es necesario primero seleccionar el Sistema Multiagente con el cual trabajar, en este caso, se ha seleccionado un "Sistema para el Dynamic Dial-A-Ride Problem (D-DARP)" que utiliza Agentes [Donoso y Sandoval, 2009] que trata sobre cómo los Sistemas Multiagente pueden ayudar a regular las rutas, los tiempos de salida, los vehículos e incluso los operadores para ajustarse a la demanda de los Sistemas de Tranporte inteligentes de pasajeros.

A continuación se realizara una breve explicación de que son estos sistemas.

5.1 Sistemas de Transporte Inteligente

El tráfico, y el transporte en general, es un tema importante en la mayoría de las ciudades grandes de los países desarrollados, en los que la congestión se ha convertido en un problema cotidiano de difícil solución, produciendo efectos indeseados en la movilidad de los conductores y peatones. El incumplimiento de los horarios en los transportes públicos, el incremento del tiempo de los viajes en el transporte público y en el privado, la contaminación ambiental y los niveles de contaminación acústica intolerables que llegan a afectar seriamente la salud, son algunos de esos efectos. Todo ello produce una disminución evidente del bienestar de la población, pero además, con lleva importantes pérdidas económicas.

Una de las respuestas más eficientes al problema de la congestión radica en el uso de sistemas informáticos y de telecomunicaciones aplicados a la gestión del tráfico. Los sistemas inteligentes de transporte están siendo un eficiente apoyo en el intento de disminuir los problemas de congestión de los transportes urbanos e interurbanos, no solo ayudando a mejorar la movilidad sino haciéndola más sostenible. Pueden ser definidos como la unión entre los avances en las tecnologías de la información y los sistemas de comunicación con los vehículos y redes de caminos que forman parte del sistema de transporte. Se pueden considerar inteligentes debido a que proveen información oportuna tanto a los usuarios como a los operadores.

Con estos sistemas se pretende resguardar la vida de las personas como así también disminuir costos de tiempo y dinero, también se ha considerado que la finalidad última de estos sistemas es la de dotar a los automóviles de la capacidad de conducir por sí mismos con lo que se reduciría el número de accidentes, se aprovecharían mejor las infraestructuras y se produciría una disminución del consumo y de la contaminación.

Esto puede lograrse a través de diversas aplicaciones de sistemas de transporte inteligentes. Estos sistemas reciben y procesan la información capturada por distintas aplicaciones, entregando soluciones globales orientadas a hacer un mejor uso de los sistemas de transporte, lo que apoya la toma de decisiones tanto de los usuarios como los operadores de la red. Aplicaciones típicas en este sentido son la localización de vehículos a través de GPS (Global Positioning System) o Sistema de Posicionamiento Geográfico y su integración en un GIS (Geographic Information System) o Sistema de Información Geográfico, sistemas de prioridad al transporte público, sistemas de información al pasajero y sistemas de regulación de frecuencias por demanda, entre otros. Entre las ventajas de su incorporación se ha constatado una importante reducción en los tiempos de viajes, principalmente para los usuarios del transporte público, un mejor uso de la flota operativa de buses y, desde el punto de vista medioambiental, un ahorro de combustible y una disminución de los contaminantes.

5.2 Sistemas Flexibles de Transporte de Pasajeros

El transporte público se enfrenta a una serie de importantes retos en todo el mundo. Durante las últimas décadas, los viajes urbanos y suburbanos han experimentado grandes cambios en términos de calidad y cantidad. Estos cambios han sido fruto de varios factores, por un lado la extensión de la urbanización ha provocado un fuerte aumento en los trayectos de las afueras a los suburbios, y largos trayectos de las afueras al centro, ha aumentado del poder adquisitivo de las persona por lo que ha habido un rápido aumento del parque automovilístico, ha habido cambios en el estilo de vida que han determinado un aumento de los viajes de ocio y de compras, que no se prestan fácilmente para el uso del transporte público [Meyer, 2003].

También se han estado produciendo cambios en los modelos de movilidad debido a que las ciudades, donde se registra un aumento de la población, generan mayores y diversas demandas de movilidad. Además, la economía actual cada vez más orientada al servicio fomenta el que las personas quieran poder elegir dentro de un abanico amplio y flexible de servicios de transporte. Los ciudadanos no sólo exigen más movilidad – es decir una movilidad más frecuente y más extendida – sino también una movilidad de mayor calidad. Ha habido un continuo incremento en el uso de vehículos particulares que ha producido un decremento en la eficiencia y uso del transporte público. Los servicios públicos de transporte no han podido responder las necesidades de transporte que requieren las personas hoy en día.

Muchas personas sienten que su movilidad se ha visto reducida porque no tienen acceso, a un costo razonable, a servicios de transporte adecuados para ellos. Existen dos problemas principales que dificultan la utilización de las personas de los servicios de transporte público, uno es el insuficiente nivel de servicio que presentan, relativo fundamentalmente a aspectos tales como frecuencia, áreas de servicio, etc., y el otro es la existencia de flotas de vehículos y rutas de vehículos inapropiadas, especialmente para las personas enfermas o discapacitadas.

Frente a las inconveniencias presentadas por los servicios de transporte tradicionales surgen los sistemas flexibles de transporte. Dentro de los cuales están los servicios de transporte de respuesta a demanda (DRT) que presentan características que son de gran utilidad dentro de la comunidad en la cual se insertan. El DRT es una forma de transporte público avanzada, orientada al usuario y caracterizada por la realización de trayectos flexibles y la programación de vehículos pequeños/medios operando en modo de trayecto compartido entre las paradas de subida y bajada, de acuerdo con las necesidades de los pasajeros.

El uso de DRT donde las rutas, los tiempos de salida, los vehículos e incluso los operadores pueden ser regulados para ajustarse a la demanda, permite un servicio orientado al usuario y reducir costos a los operadores, a la sociedad y a los pasajeros.

Este tipo de transporte constituye un complemento a la actual red de transporte, de hecho se habla de que constituye una solución de transporte *intermedia* entre los servicios de transporte público y los proporcionados por los radio taxis. En efecto, este tipo de transporte está destinado a suplir las necesidades de transporte de personas con necesidades de transporte especiales como lo son los enfermos, discapacitados o los adultos mayores, así como también de suplir la demanda de transporte en épocas de baja demanda, como la que se produce en los días feriados, fines de semana o en las noches y también entregar servicios de transporte en poblados con baja población [Cubillos, 2004].

Para llevar a cabo la implementación y manejo de estos servicios, es necesario que los usuarios de los servicios de transporte de respuesta a demanda cuenten con instrumentos *amigables* para acceder a los servicios y que la organización encargada de los servicios sea capaz de manejar un conjunto de recursos de transporte, relacionados de manera dinámica, de manera de adaptar adecuadamente los servicios a la demanda. Para esto, es necesario que la arquitectura de los sistemas de respuesta a demanda cuente con adecuada tecnología de comunicaciones y herramientas tecnológicas [Cubillos, 2004].

5.3 Dial-A-Ride Problem

Los sistemas de transporte de respuesta a demanda, desde el punto de vista de la investigación de operaciones y de las matemáticas, corresponden a un problema conocido en la literatura denominado como Dial-A-Ride Problem (DARP). En dicho problema, los usuarios formulan pedidos de transporte desde un origen específico (punto de recolección) a un destino específico (punto de entrega). La clave del problema es la de diseñar un conjunto de rutas de vehículos de mínimo costo que se acomode a todas las peticiones de transporte hechas. Existen diversas restricciones que debe satisfacer la solución planteada, entre ellas se pueden nombrar aquellas relativas a la

capacidad, duración, ventanas de tiempo, precedencia, tiempo de viaje, entre otras, también se hacen restricciones relativas al tipo de servicio deseado como es el uso compartido o exclusivo del vehículo, el uso de espacios para sillas de ruedas y cualquier otro servicio complementario [Cordeau, 2003].

Desde el punto de vista del modelado, el DARP generaliza un número de problemas de encaminamiento de vehículos tales como el PDVRP (Pickup and Delivery Vehicle Routing Problem) y el VRPTW (Vehicle Routing Problem con Time Windows). Lo que hace diferente al DARP de esos problemas de enrutamiento de vehículos, es la perspectiva humana, debido a que cuando se trata de transportar pasajeros, la reducción de las inconveniencias al usuario debe ser equilibrada con la reducción de los costos de operación. Además es importante señalar, que la capacidad del vehículo constituye una restricción en el DARP, mientras que esa restricción es poco relevante en otros problemas de PDVRP, particularmente en los problemas relativos a la recolección y entrega de cartas y paquetes pequeños [Cordeau, Laporte, Ropke, 2006].

Los servicios de DARP pueden operar de un modo estático o dinámico. En el caso estático, todos los pedidos de transporte son conocidos de antemano, mientras que en el caso dinámico, los pedidos de transporte son conocidos gradualmente durante el día, por lo que las rutas de los vehículos deben ser ajustadas en tiempo real de acuerdo a la demanda. De ahí se derivan las dos versiones del DARP, conocidas con el nombre de S-DARP (Static Dial-a-Ride Problem) y D-DARP (Dynamic Dial-a-Ride Problem) [Cordeau, 2003]. Cabe señalar que la versión dinámica del problema se puede reducir a un conjunto de peticiones estáticas sucesivas, por lo que, las aproximaciones de solución que existen para la versión estática, pueden aplicarse a la versión dinámica del problema mediante la forma descrita.

En el Sistema seleccionado se considera la versión dinámica del problema, es decir, se aborda el Dynamic Dial-a-Ride Problem. Además, se considera un ambiente dinámico, en el cual se monitorea el progreso del vehículo, también se considera que los clientes pueden modificar o cancelar sus peticiones de transporte, pueden ocurrir retrasos de vehículos, los clientes pueden no llegar al lugar de recolección y los vehículos pueden fallar, eventos que implican la reprogramación de los viajes y su administración.

5.4 Arquitectura Multiagente de Transporte MADARP

De un conjunto de arquitecturas disponibles para problemas de transporte, la utilizada en este Sistema es MADARP **[Cubillos, 2004]**. Esta es una arquitectura destinada a planificar y programar los pedidos de transporte en un ambiente dinámico, dentro del contexto de los sistemas de transporte de pasajeros. La arquitectura proporciona un sistema base de agentes que realizan la interfaz básica, el planeamiento y los servicios de soporte para manejar diversos tipos de peticiones de transporte, usando una flota heterogénea de vehículos.

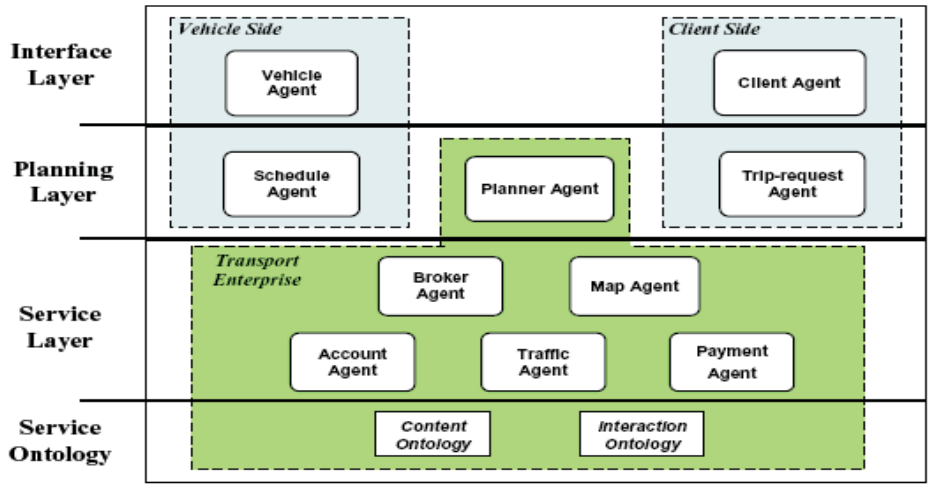

Ilustración 5-1 Arquitectura de Agentes MADARP [Cubillos, 2004]

En la Ilustración 5-1 se puede apreciar la arquitectura de agentes MADARP, la descripción de la arquitectura se hace en base a [Cubillos, 2004]. En el modelo se pueden apreciar dos perspectivas, en la primera de ellas se muestra a los agentes subyacentes agrupados en torno a capas funcionales destinados a proporcionar un servicio coherente. Mientras que la otra perspectiva, muestra la arquitectura desde el punto de vista de los agentes involucrados, identificando los agentes referentes a cada actor del sistema.

Existen cuatro capas en la arquitectura que agrupan a los agentes y las estructuras de acuerdo a la funcionalidad proporcionada. A continuación se hace una breve descripción de la funcionalidad de cada capa:

- *Capa de interfaz*: Conecta al sistema con el mundo real, proporcionando agentes capaces de conectarse con los actores del sistema (clientes, operadores de vehículos).

- *Capa de planeamiento*: Contiene los agentes dedicados a realizar el procesamiento y planeamiento de los viajes de una manera distribuida.

- *Capa de Servicio*: Apoya a la capa de planeamiento proporcionando diversas funcionalidades complementarias, necesarias para manejar un servicio de transporte integral.

- *Servicio de Ontología*: Proporciona medios de integración y cooperación de los diversos agentes y actores de las capas superiores de una manera transparente y coherente.

Cabe señalar, que el control en esta arquitectura, está distribuido entre las diferentes capas. En términos generales, los agentes de interfaz proporcionan la entrada y la señal monitoreada para que los agentes de planeamiento puedan ajustar la planificación de los vehículos. Los agentes de servicio apoyan estos procedimientos proveyendo la información requerida para el proceso de replanificación y la ontología ofrece los conceptos y formalización para llevar a cabo el control de interacciones.

Los tres actores principales involucrados en la cadena del transporte son los vehículos, los clientes, y la empresa de transporte. Cada uno de ellos se encuentra modelado en términos de agentes. Consecuentemente, cada actor vehículo es representado por un agente Vehículo y un agente Schedule. De una manera similar, cada cliente es caracterizado por un agente Cliente y por un agente Trip-request. En ambos casos, el par de agentes se encuentran altamente acoplado porque ellos están modelando diferentes aspectos de la misma entidad real. El tercer actor es la empresa de transporte, que está construido sobre una serie de agentes y estructuras que proveen soporte a diversos servicios relacionados con la planificación y control del servicio de transporte de pasajeros proporcionado.

Los clientes individuales y sus requerimientos son capturados por los agentes Cliente y Trip-request, juntos proveen la total comunicación e interoperabilidad del usuario final real con el sistema de transporte. El agente cliente esta encargado de proporcionar una interfaz de usuario personalizada mientras el Trip-request maneja el proceso de pedir el servicio, su característica y la decisión siendo requerida.

El agente cliente es responsable de capturar todos los requerimientos de los clientes no solo los concernientes al tipo de transporte deseado sino que también sus preferencias sobre situaciones de contingencia (por ejemplo, atrasos, congestión vehicular, desviaciones, etc.). El agente Trip-request toma esos requerimientos y preferencias para actuar a favor del cliente durante todo el proceso. Dependiendo del grado de autonomía proporcionado por el cliente, el agente Trip-request puede actuar como un asistente de viajes personal o simplemente como un mero intermediario de las decisiones del cliente.

Cada vehículo real es representado por una pareja de agentes, el agente Vehículo y el agente Schedule. Ellos proporcionan la interoperabilidad entre los vehículos que ellos representan y el sistema de transporte al cual pertenece. El agente Vehículo representa un rol de interfaz, proporcionando al vehículo y su conductor un canal de comunicación durante la jornada acerca de cualquier contingencia que pudiera presentarse. Por otra parte, el conductor recibe a través del agente Vehículo cualquier información relativa a cambios en la planificación original como un nuevo cliente para transportar, cancelaciones de clientes y cambios relevantes en las condiciones de tráfico entre otras. El agente Schedule está encargado de manejar la ruta del vehículo y procesar cualquier nuevo pedido de transporte de un cliente.

El agente Vehículo también es el responsable de monitorear e informar acerca del estado de los vehículos y el progreso del servicio a través del día. De una manera complementaria, el agente Schedule es responsable de realizar cualquier ajuste en la ruta de los vehículos o en la programación de los viajes, motivado por cualquier cambio o eventualidad. Ambos agentes dependen uno del otro debido a que el primero actúa como una especie de sensor de que está sucediendo en el ambiente y en el vehículo, mientras que el segundo razona basado en las entradas y toma las decisiones con respecto a la planificación que entonces las comunica al vehículo real a través del agente vehículo.

El rol de servicio de transporte se realiza principalmente por el agente Planner que actúa como cara visible de los clientes. Además, hay un sistema entero de agentes que colaboran para dar soporte a las diferentes funciones requeridas, como por ejemplo concordar las peticiones con los vehículos, el acceso a datos geográficos, la contabilidad de transacciones y el servicio de pago entre otros. De ellos, los agentes más críticos para el punto de vista de la planificación y control son el agente Broker y el agente Map.

6 DESARROLLO DEL SISTEMA

6.1 Analisis Sistema Existente

Para comenzar el Desarrollo, es necesario analizar el Sistema Multiagente, y determinar cuáles son las partes a modificar para lograr la integración de los nuevos Agentes encargados de las graficas 3D. Por esto se realizó el análisis del Sistema tomando el diagrama Passi de Identificación de Agentes, que viene a ser el mismo que de Descripcion del Dominio donde se muestran todas las funcionalidades ofrecidas por el sistema, sólo que ahora se agrupan dependiendo del agente responsable de ellas.

6.1.1 Diagrama de Identificación de Agentes

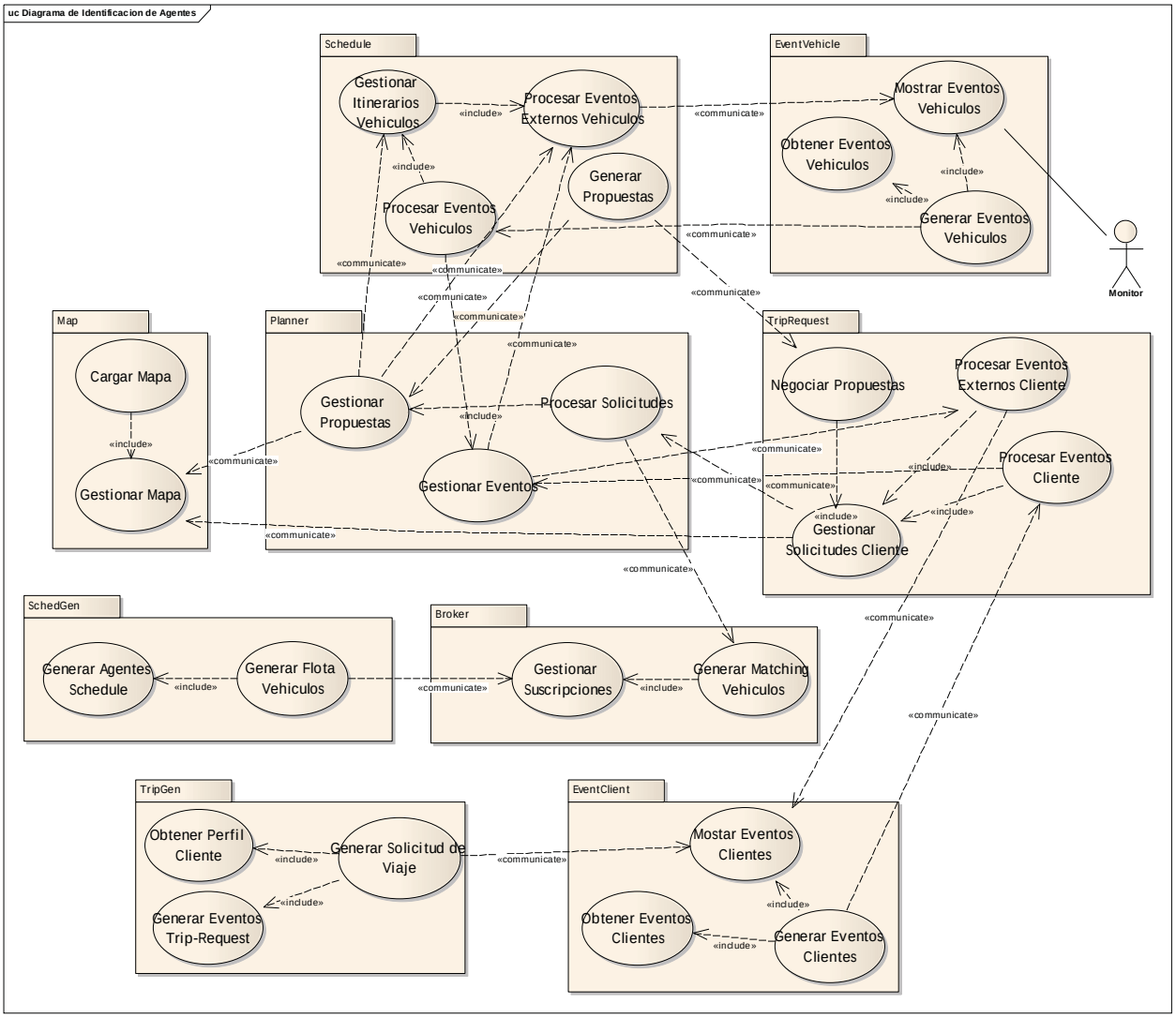

Ilustración 6-1 Diagrama de Identificación de Agentes

De este diagrama poder identificar claramente cuáles son las zonas de importancia para este trabajo, si consideramos que toda la lógica resolutiva del sistema respecto a la toma de decisiones no es relevante, sino los puntos extremos donde las instrucciones a cada agente vehículo son enviadas, indicando su próxima acción, el punto donde esto se realiza es en el Agente EventVehicle, puntualmente en “Generar Eventos Vehículos”

6.1.2 Diagrama de Identificación de Agentes + Visualizador 3D

A continuación se presenta el mismo Diagrama de Identificación de Agentes, al cual se le ha agregado el nuevo Agente encargado del entorno 3D (Manager3D) y la coordinación respecto al Sistema Multiagente.

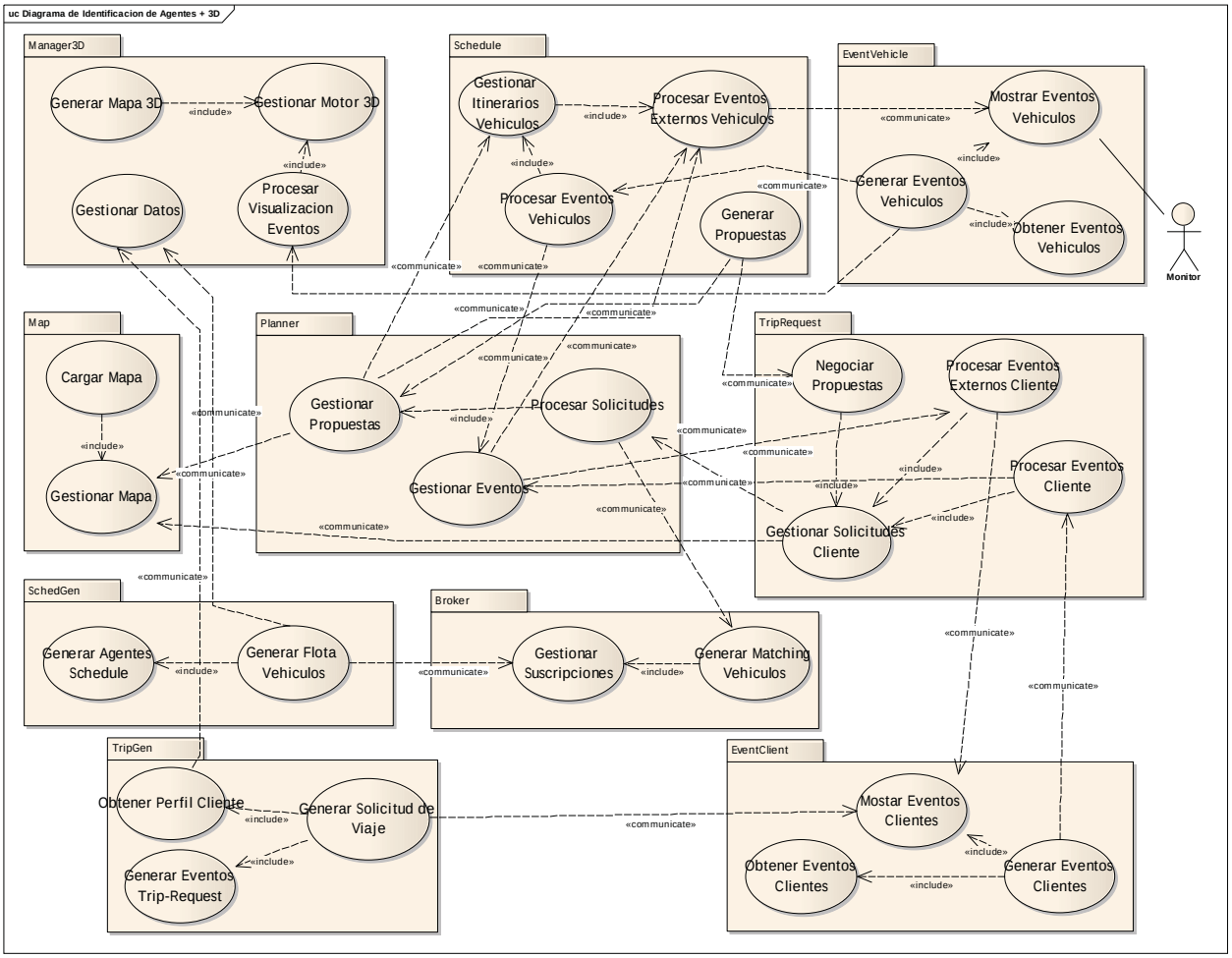

Ilustración 6-2 Diagrama de Identificación de Agentes + Manager3D

Podemos observar cómo en la Ilustración 6-2 se agrega el nuevo Agente Manager3D que se encarga de todo lo relacionado con el entorno 3D, como es la generación del Mapa 3D, la creación de los Vehículos y Clientes en 3D, la actualización de los eventos en el entorno 3D, etc. También apreciamos como el Manager3D recibe la información inicial de los Vehículos desde el Agente SchedGen que es el encargado de crearlos, por otro lado el Manager3D recibe todos los usuarios desde el Agente TripGen que se encarga de crear cada uno de los usuarios según los datos entregados en el txt de entrada. Finalmente, la actualización de los eventos realizados por los vehículos es informada al Agente Manager3D por el Agente EventVehicle.

6.2 Desarrollo Integración

Actualmente el Sistema Multiagente utiliza un archivo “Adartw.ini” en el cual se encuentran los parámetros que utilizará, como son la ubicación y nombre de los archivos con los datos de entrada, diversos datos utilizados por el “solver” del sistema, etc. Este mismo archivo se modificará para indicar al sistema si se desea o no utilizar la visualización en 3D y en caso de ser así, la ubicación del archivo donde se detalla la distribución de los diferentes nodos involucrados para la creación del mapa.

```
MODE = 0
A = 0 0 3 0
B = 1.10
WS = 0 0 2 0
DIRECTORY = ppp
C1 = 1.0
C2 = 0.0
C3 = 1.0
C4 = 0.0
C5 = 1.0
C6 = 1.0
soon = true
graph_check = true
COMPRESSED_MATRIX = false
runs = 1
GRAPH_FILE = 5C.txt
VEHICLES_FILE_M0 = BUSES5.txt
3D_VIEWER = 1
3D_FILE  = 3D.txt
USERSFILE_M0 = U
USERSSTART_M0 = 1
USERSFILECOUNT_M0 = 2
VEHICLES_FILE_M1 = buses.txt
USERSFILE_M1 = P
USERSSTART_M1 = 1
USERSFILECOUNT_M1 = 2
```

Ilustración 6-3 Formato Archivo "Adartw.ini"

Los nuevos parámetros son 3D_Viewer y 3D_File , donde el primero puede tener los valores 1 o 0 , indicando si se desea o no visualización en 3D respectivamente, y el segundo la ubicación del archivo txt que indique la distribución en 3 dimensiones de los nodos involucrados.

1	1	10	0
2	1	1	0
3	5	5	0
4	5	1	0
5	9	4	0
6	9	8	0
7	13	11	0
8	12	7	0
9	12	1	0

Ilustración 6-4 Formato Archivo 3D_File

En la Ilustración 6-4 se observa el formato requerido para el archivo indicado en el parámetro 3D_File donde en la primera columna se encuentra un String de identificación de cada nodo, y en las siguientes columnas su posición en el eje X,Y y Z respectivamente.

Actualmente, el sistema se encuentra cargando el motor 3D a demanda según lo especificado en el archivo Adartw.ini, cargando los diferentes nodos en un ambiente completamente 3D. Los diferentes modelos 3D utilizados fueron creados utilizando la herramienta de modelado 3D Sketchup [Sketchup] .

A continuación, se muestra en la Ilustración 6-5 la ventana principal del sistema, en ella se pueden apreciar el entorno creado para los Vehículos, así como la distribución en base al mapa entregado en el archivo .txt de configuración de los Paraderos (nodos). Al costado derecho de visualiza la barra de menú donde se es posible seleccionar desde un listado un bus a seguir durante su recorrido, así como cambiar entre las distintas cámaras predefinidas.

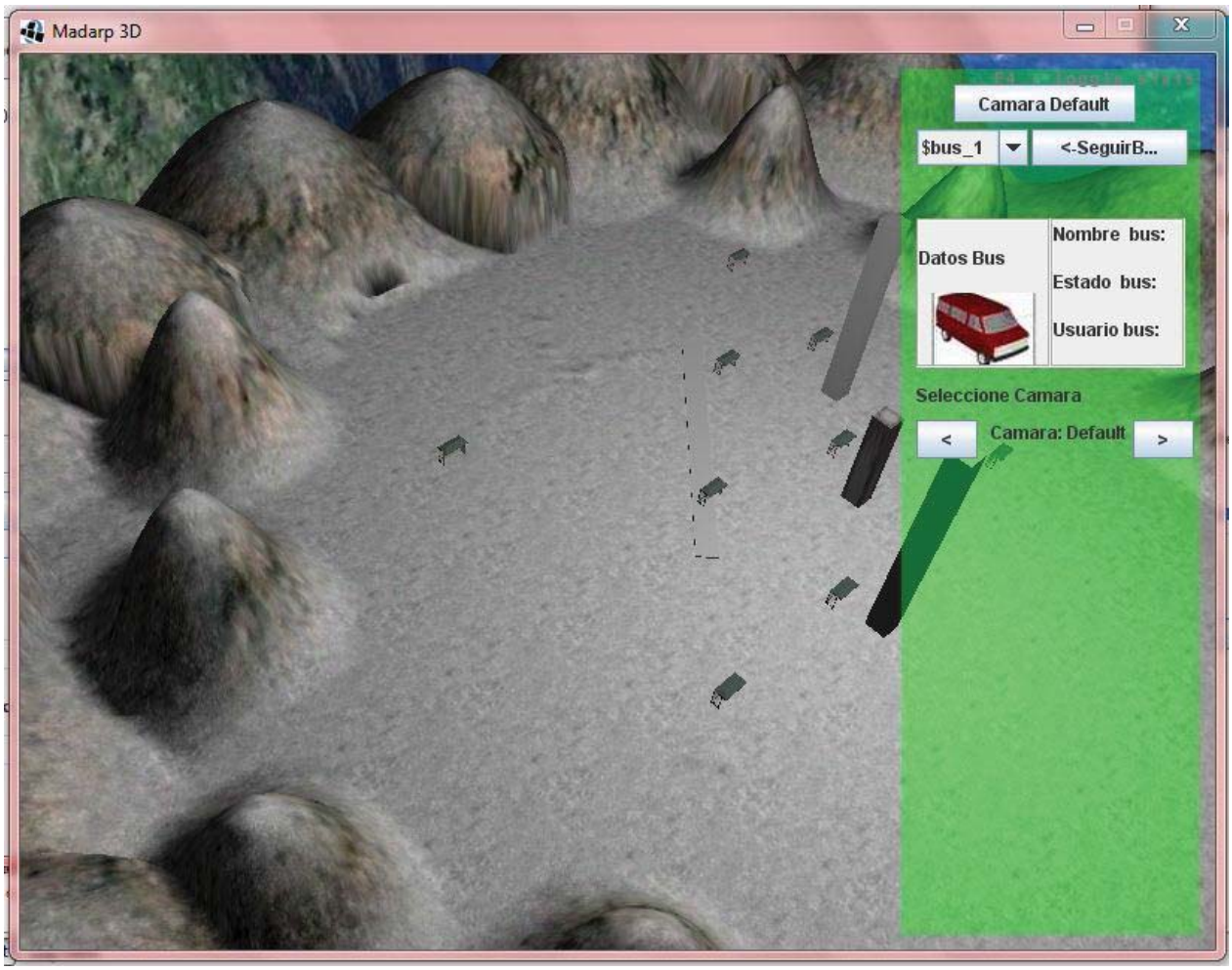

Ilustración 6-5 Ventana Principal de la Visualización 3D

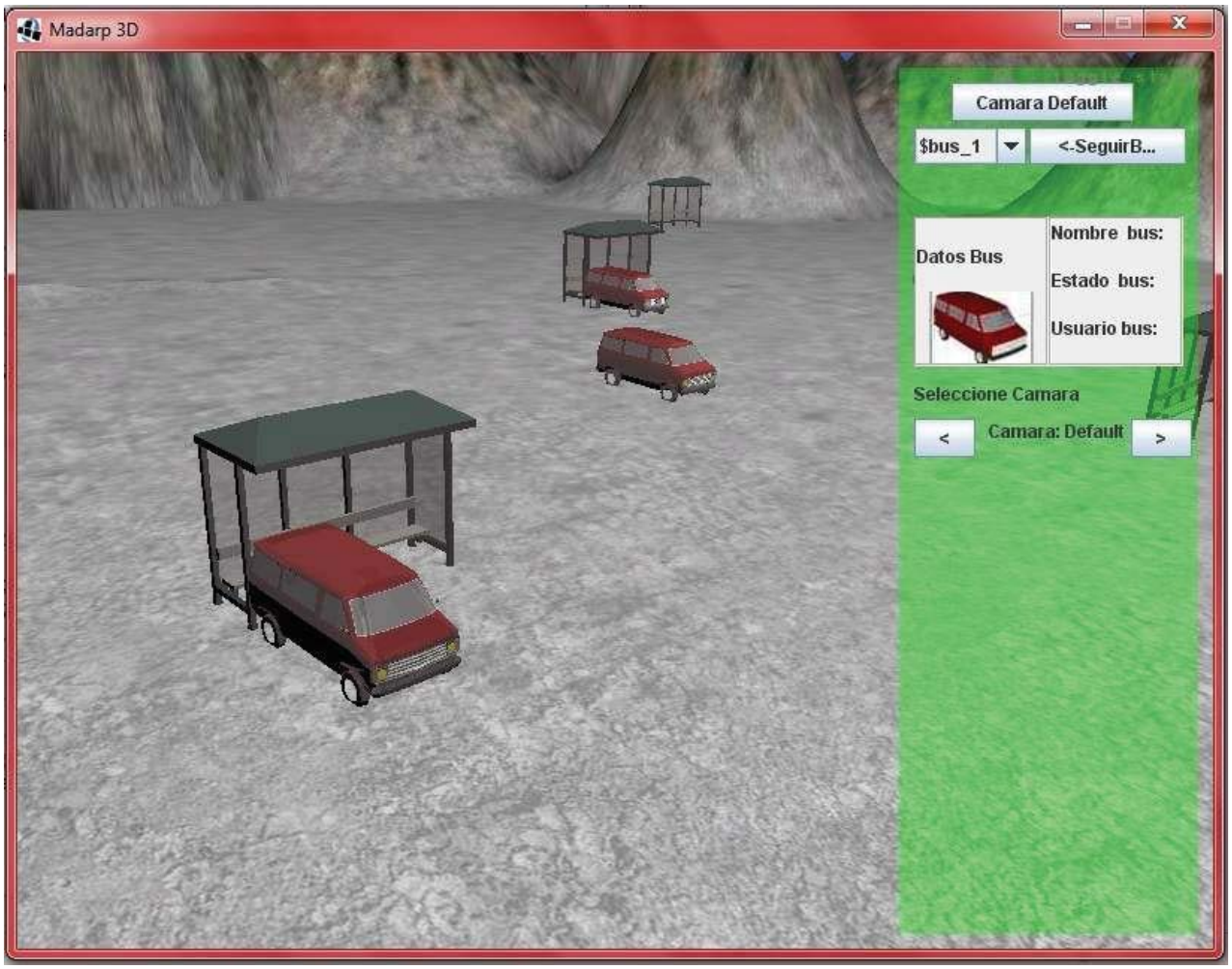

Ilustración 6-6 Primer Plano Bus y Parada en el Sistema

También es posible ver al lado derecho en la Ilustración 6-6 un recuadro con la imagen de un Bus, en este podemos ver la información de cualquiera de los buses, como es el nombre, el estado en que se encuentra y los pasajeros en él, todo esto con solo presionar sobre uno de ellos.

La Ilustración 6-7 presenta la consola principal de control del MADARP, desde ella es posible ver el estado de cada uno de los vehículos, el desplazamiento de estos, etc. También entrega toda la información correspondiente a los eventos que suceden como puede ser la cancelación de una solicitud de transporte, etc. Es importa destacar que esta era la única forma desplegada por el sistema al momento de ejecutarlo, debido a esto se buscó mejorar la visualización de esta información implementando la interfaz 3d donde se pudiera apreciar más fácilmente el desplazamiento, así como también el despliegue realizado por el grupo de vehículos simulados en el sistema.

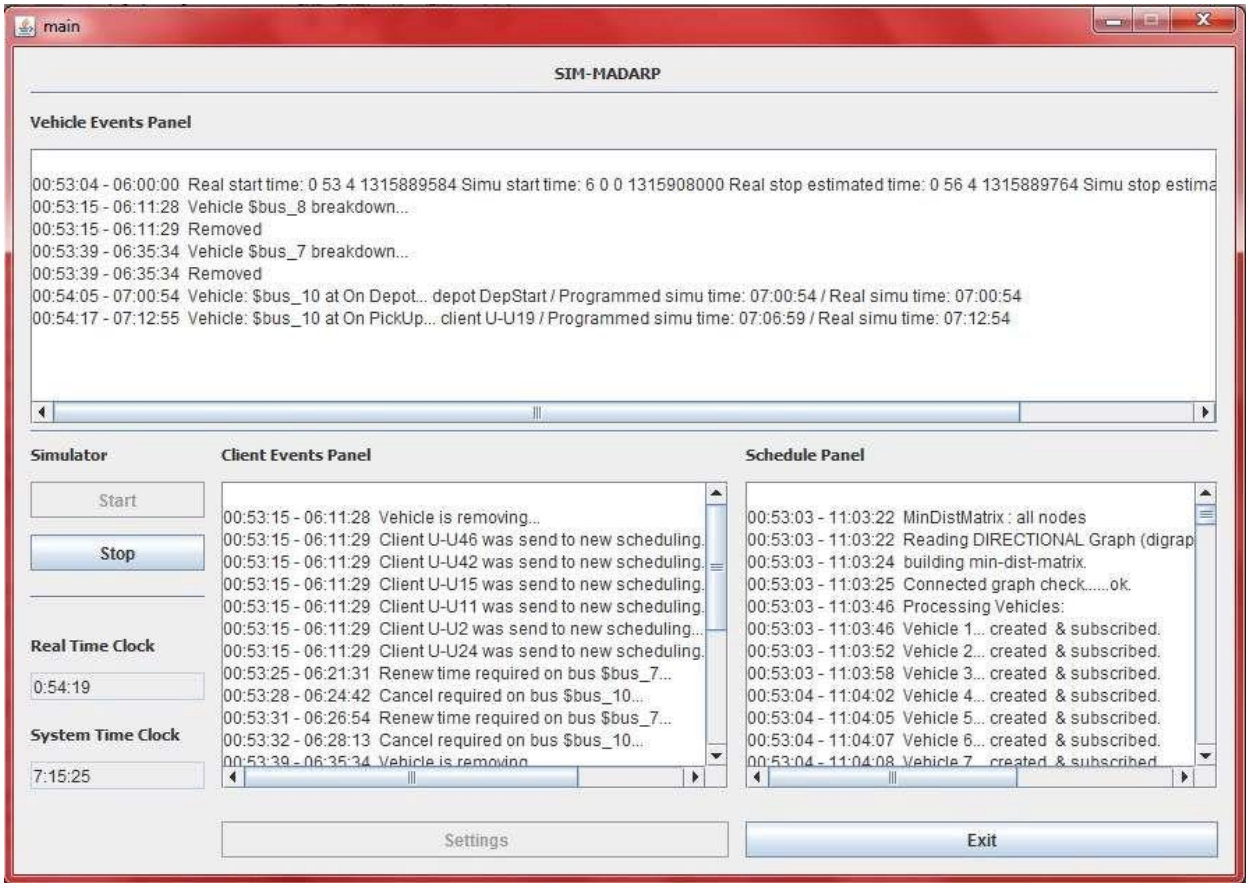

Ilustración 6-7 Consola Principal del MADARP

6.2.1 Diagrama de Casos de Uso Interfaz 3D

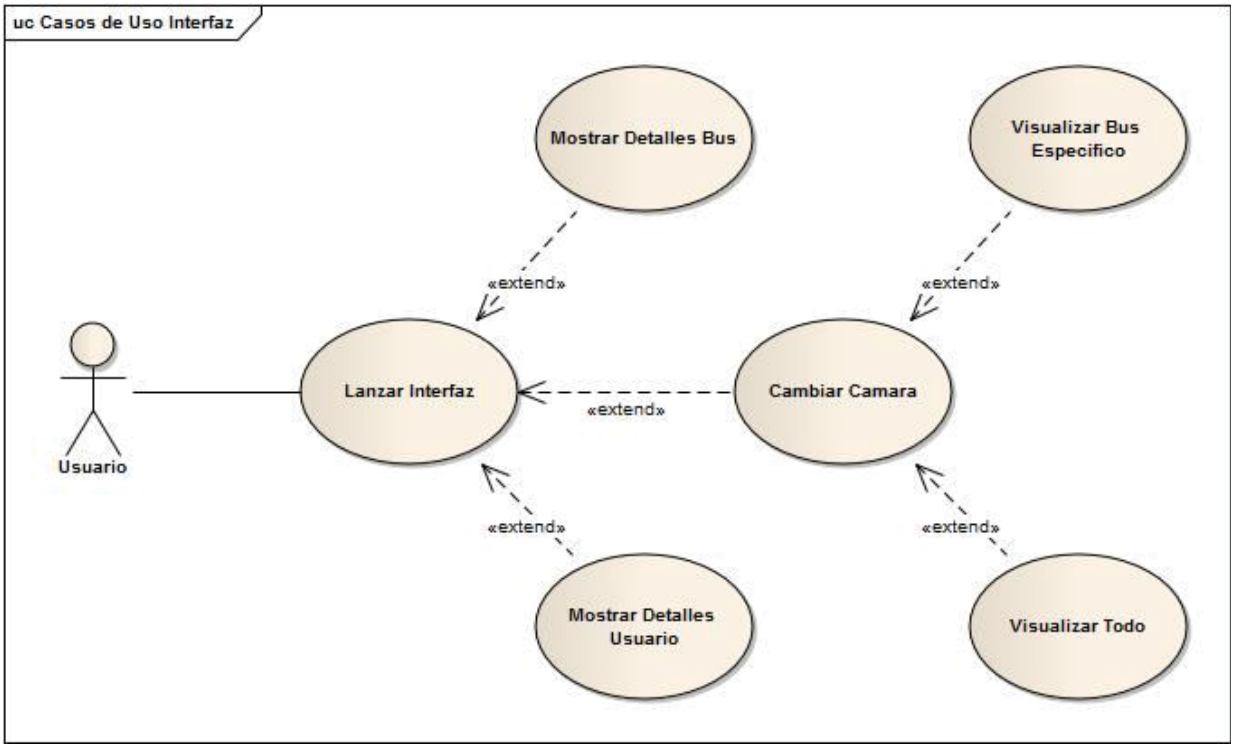

Ilustración 6-8 Diagrama de Casos de Uso Interfaz 3d

En el diagrama de la Ilustración 6-8 se pretende mostrar las diferentes funcionalidades que la interfaz 3d entrega al usuario, entre ellas se encuentran las diferentes opciones de cámara, además de permitir ver en detalle la información de cada vehículo.

6.2.2 Funcionamiento del Motor Gráfico 3D, Diagrama de Actividades

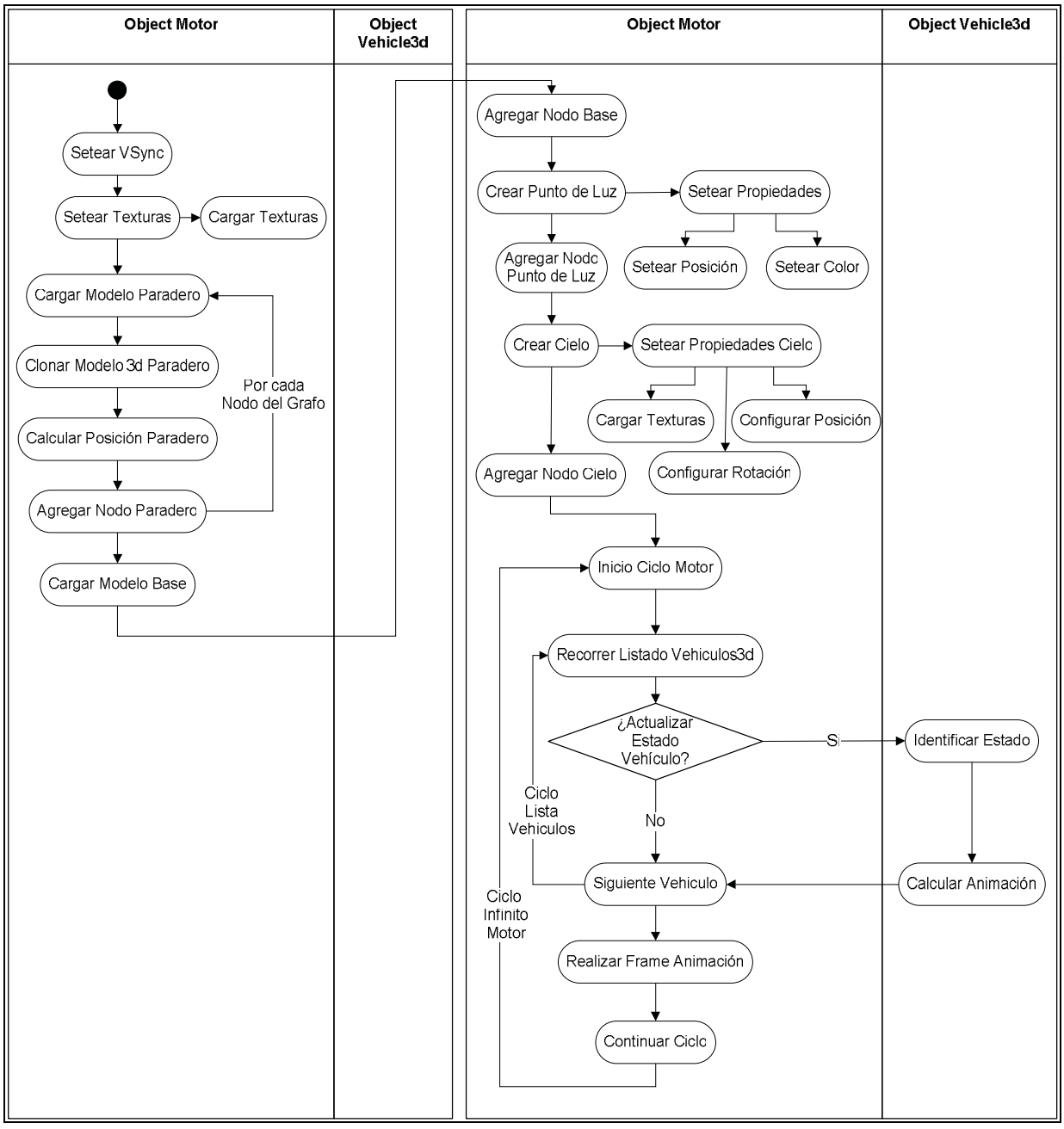

Ilustración 6-9 Diagrama de Actividades, Funcionamiento Motor Gráfico

En la Ilustración 6-9 se aprecia un diagrama de actividades en el cual se detalla cómo funciona el motor gráfico, desde su inicio, hasta el ciclo infinito que posee.

En un comienzo el motor realiza sus configuraciones, como son la activación del VSync para mejorar la calidad de la imagen desplegada, realiza la carga de todas las texturas que se utilizaran en los modelo, etc. Posteriormente carga el modelo de los Paraderos, y lo clona, de forma de no realizar la conversión del Modelo desde el formato de origen (OBJ) al manejado por JMonkey (JME) por cada vez que se necesite un nuevo paradero, sino que se solicita un clon del original, luego calcula la posición donde se debe ubicar, y lo agrega al nodo principal. Todo esto lo hace por cada uno de los paraderos existentes.

Luego carga el modelo de la base y lo agrega al nodo también, posteriormente crea el punto de luz del sistema, asignándole el color a la luz y el origen de la misma. A continuación se genera el cielo, que es lo que se ve a lo profundo en el sistema, se cargan las texturas correspondientes, se configura su posición y rotación, de forma que coincidan con la posición del resto de los modelos, además es necesario configurar la cámara, para que siga a la base que es nuestro punto de referencia, finalmente se agrega al nodo principal.

Todo motor gráfico consta de un ciclo infinito, dentro del cual se ubican las operaciones que debe estar continuamente calculando el motor, como pueden ser el control de los modelos, condiciones varias, control de colisiones, o como en este caso la animación de desplazamiento de los diferentes vehículos por el entorno. Para lograr esto, por cada ciclo se recorre la lista de vehículos 3d, buscando si existe algún desplazamiento que realizar, de ser así, dentro del objeto que representa al vehiculo3d, se realiza el cálculo del desplazamiento, comenzando por la ubicación de los puntos a los cuales se moverá (origen y destino), se genera el vector que une ambos puntos, se divide el largo por un valor que determina la velocidad a la cual se desplazara, y se guarda dicho valor, esto se hace por cada vehículo del arreglo.

Posteriormente se actualiza la posición de cada vehículo, sumando la fracción calculada del vector, una vez por cada ciclo.

6.2.3 Diagrama de Clases Paquete Graphics3D

En el siguiente diagrama (Ilustración 6-10) se muestran las nuevas clases de paquete graphics3D que se ha integrado al Sistema original.

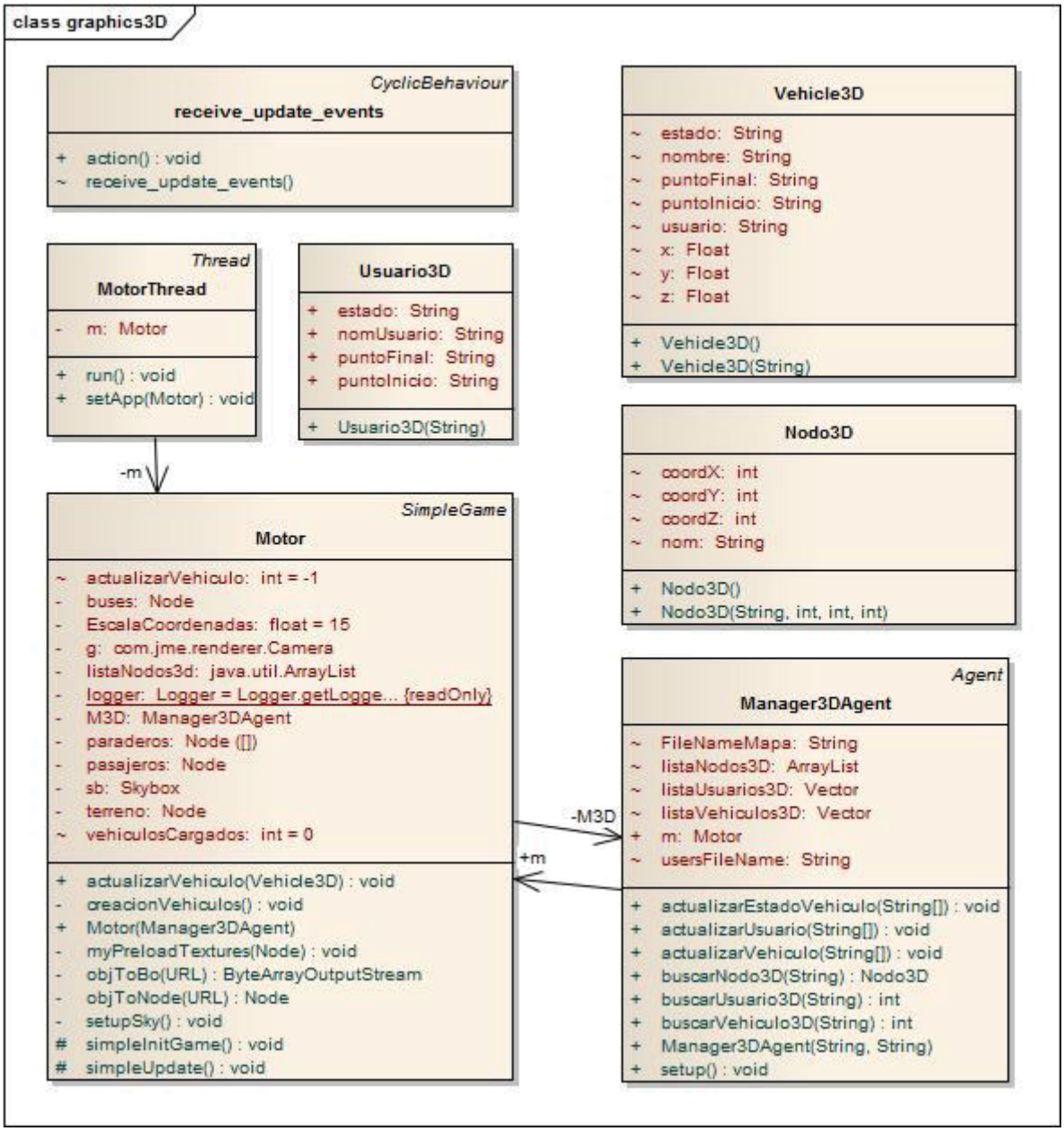

Ilustración 6-10 Diagrama de Clases paquete graphics3D

Se han creado una serie de nuevas clases, la primera llamada Motor es la relacionada directamente con el motor gráfico, es la que lanza la interfaz, maneja las variables, carga los modelos 3D, actualiza sus posiciones en el plano 3D, etc., también está la clase Nodo3D, en ella se guardan cada uno de los nodos involucrados en la interfaz 3D, estos son cargados desde el archivo txt de entrada identificado en el parámetro 3D_File del archivo de configuración.

En las clases Usuario3D y Vehiculo3D se guardan los datos de los usuarios y los vehículos respectivamente, toda esta información es la que recibe directamente el agente Manager3D desde los otros agentes. También se encuentra la clase MotorThread que permite lanzar el motor 3D en un hilo de ejecución independiente. Finalmente, se encuentran las clases Manager3DAgent y receive_update_events, una es el Agente Manager3D y la otra el behaviour que permite la recepción de los mensajes por el nuevo agente. La clase Manager3DAgent maneja una lista completa de todos los nodos3D que existen (paraderos), de todos los vehículos y usuarios a visualizar.

6.2.4 Diagramas de Identificación de Roles

A continuación se presenta el diagrama de Identificación de roles de Agentes, específicamente el del agente Manager3D, que muestran que acciones realizan en el momento de iniciar la visualización gráfica.

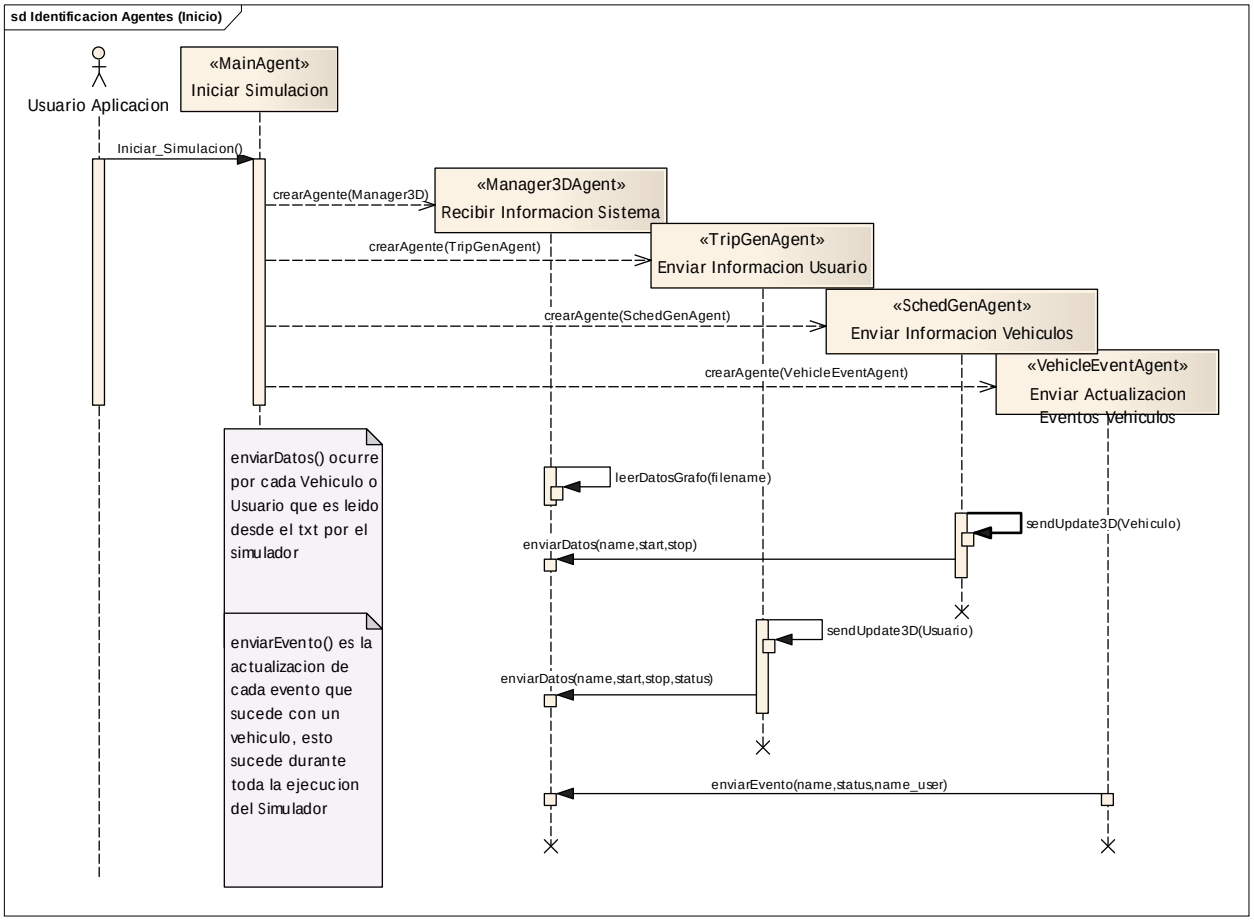

Ilustración 6-11 Identificación de Roles Inicio Visualización Gráfica

En la Ilustración 6-11 se puede observar como comienza la Visualización Gráfica, en primer lugar es el usuario el que da el inicio, esto crea al MainAgent que crea cada uno de los agentes involucrados, en este caso solo se muestran los que son relevantes para este trabajo, el nuevo Manager3DAgent,el TripGenAgent, el SchedGenAgent y el VehicleEventAgent. A continuación el Manager3DAgent realiza la carga de los datos del plano3D desde el archivo correspondiente indicado en el archivo de configuración inicial. Posteriormente el Manager3DAgent recibe por mensajes los diferentes vehículos creados desde el SchedGenAgent, por cada vehículo que este agente crea, envía un mensaje con esos datos al Manager3DAgent, que lo recibe y almacena en una arreglo propio, de la misma forma recibe los datos de los usuarios desde el TripGenAgent.

En diagrama también se quiere mostrar cómo se realiza la actualización de los eventos de los vehículos; cuando el VehicleEventAgent genera un nuevo evento para un vehículo determinado, se envía un mensaje al Manager3DAgent con los datos (nombre vehículo, estado del evento, usuario del evento). Este mensaje es recibido y se analizan localmente en el agente la lista de los vehículos, usuarios y ubicaciones en el plano 3D, para determinar la posición del bus, que acción debe realizar, y donde se debe dirigir.

6.2.5 Diagrama de Especificación de Tareas

A continuación se presenta el Diagrama de Especificación de Tareas de los Diferentes Agentes involucrados.

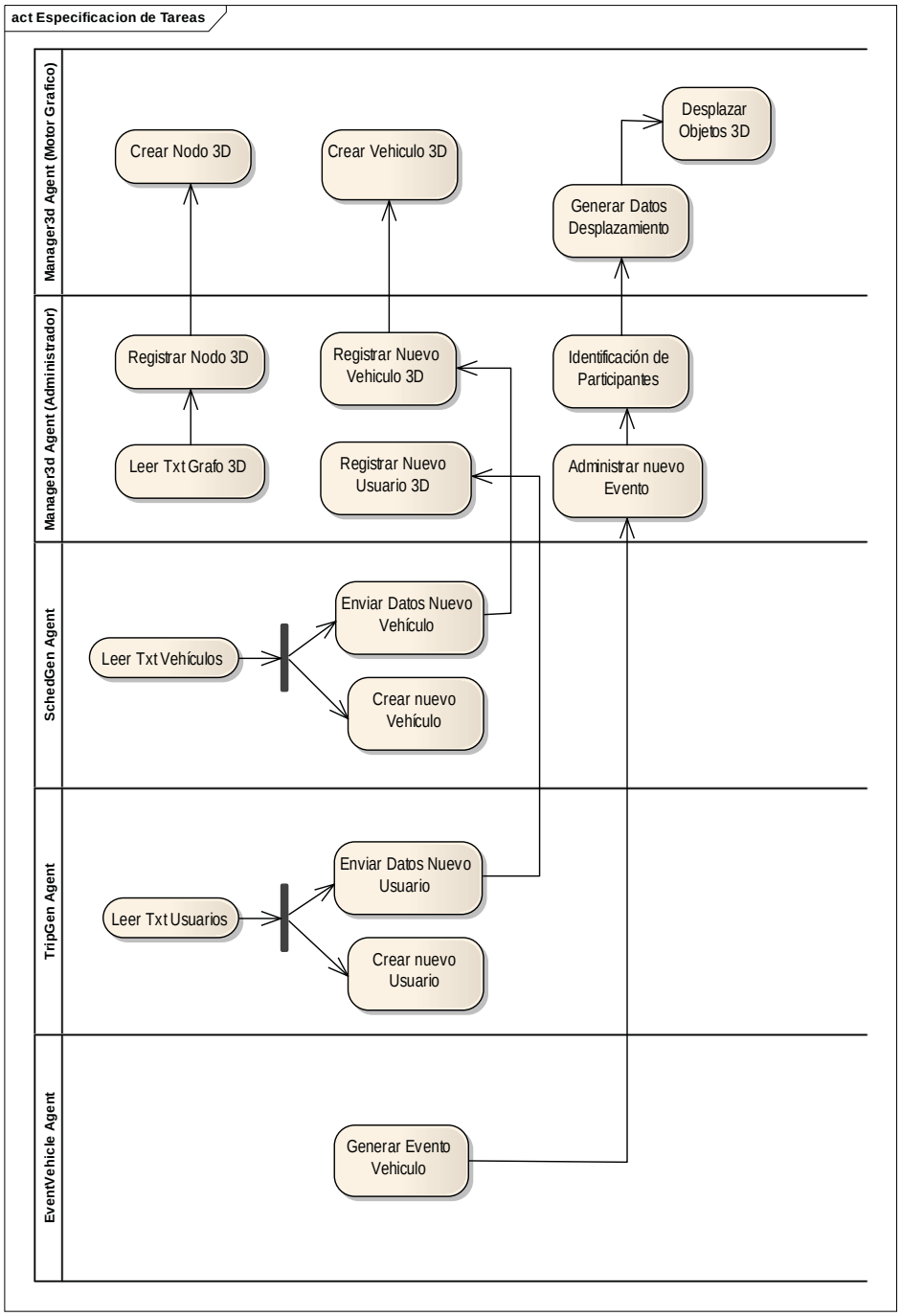

Ilustración 6-12 Diagrama de Especificación de Tareas

En la Ilustración 6-12 apreciamos todas las tareas relacionadas con el Agente Manager3d, desde que comienza con la carga del grafo3d desde un archivo, hasta cuando recibe los distintos elementos desde los demás agentes (EventVehicle, TripGen, SchedGen), además de distinguir que tareas son las correspondientes a cada uno de los roles que este Agente posee.

6.2.6 Diagrama de Descripción de Roles

El Diagrama presentado a continuación es el de descripción de Roles.

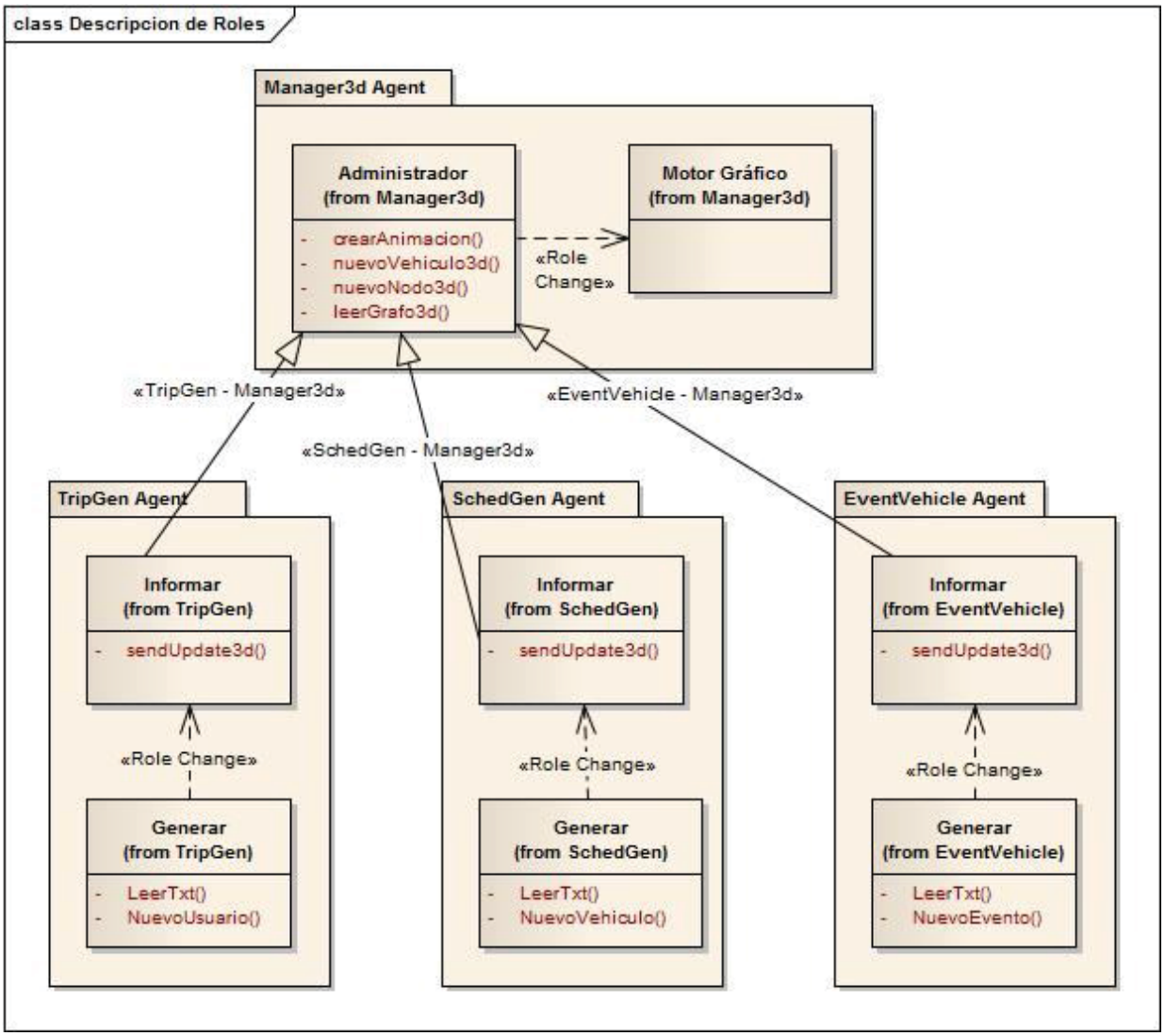

Ilustración 6-13 Diagrama de Descripción de Roles

En la Ilustración 6-13 se puede apreciar como el agente Manager3d posee 2 roles principales dentro del sistema, uno relacionado con la administración de toda la información proveniente de los otros agente involucrados, y otro referente a todo el manejo del Motor Gráfico 3d, como son las animaciones, y el manejo de los diferentes modelos utilizados.

Además, se pueden ver los otros 3 agentes que interactúan con el nuevo agente: EventVehicle, TripGen, SchedGen. Todos estos poseen una serie de roles, pero de los que nos interesa diagramar son 2 por cada uno, su rol de generador de los diferentes elementos (usuario, vehículo) y eventos. Cada uno de estos datos es enviado al Manager3d mediante el rol de "informar" que cada uno posee.

6.2.7 Diagrama de Ontología de Comunicación Passi

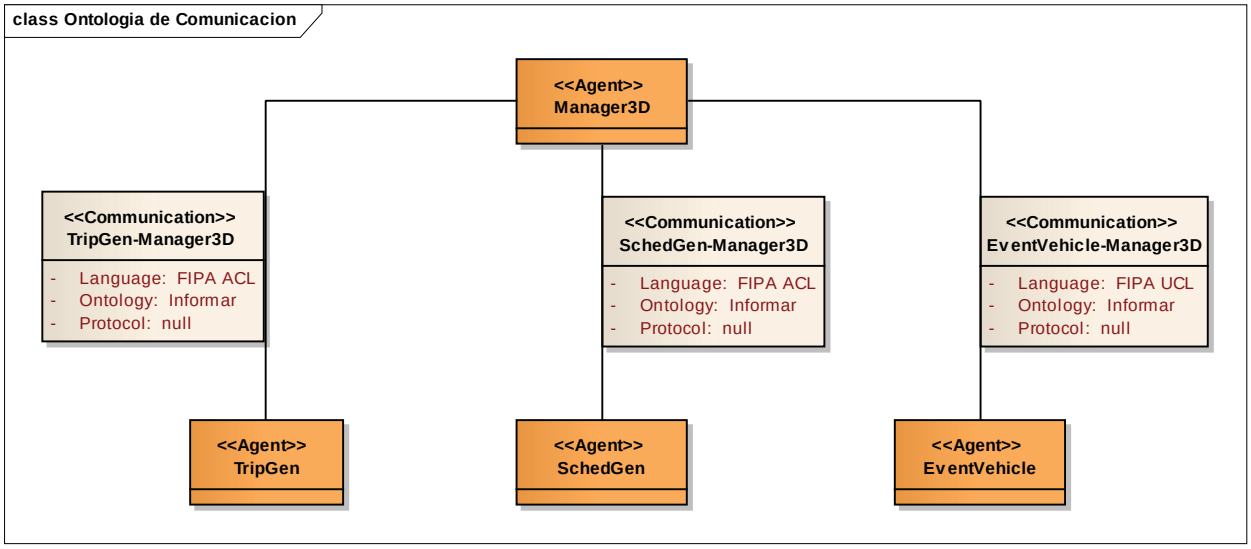

Ilustración 6-14 Diagrama de Ontología de Comunicación Passi

En la Ilustración 6-14 que se refiere a la Ontología de Comunicación de Passi podemos ver la comunicación de los diferentes Agentes, así como el lenguaje utilizado por cada uno para comunicarse con los demás. Para la comunicación de todos se usara FIPA ACL.

6.2.8 Diagrama de Estructura de Agente (Manager3d Agent)

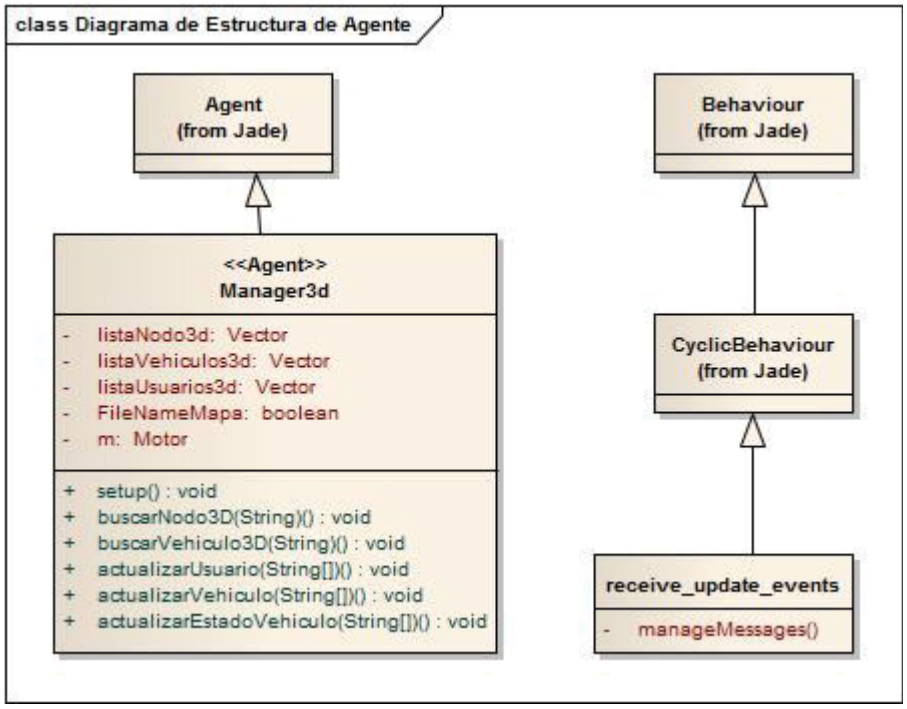

Ilustración 6-15 Diagrama de Estructura de Agente (Manager3d Agent)

En la Ilustración 6-15 se especifica al Agente Manager3d y se muestran con detalle todos los atributos que posee, además de las operaciones que realizara dentro del sistema. Podemos observar como el Agente mantiene una lista propia con cada uno de los elementos a graficar, como son los vehículos, además de la ubicación de cada uno de los usuarios, y de la distribución de los anteriores.

6.3 Modelos Gráficos

En la visualización del sistema se pretende mostrar diferentes elementos que existen en él. Para esto se diseñaron 2 modelos 3d utilizando la herramienta de modelado para Gráficos 3d Sketchup [Sketchup]. Estos modelos son los que representan a los Vehículos y a las Paradas (Parada de Bus)

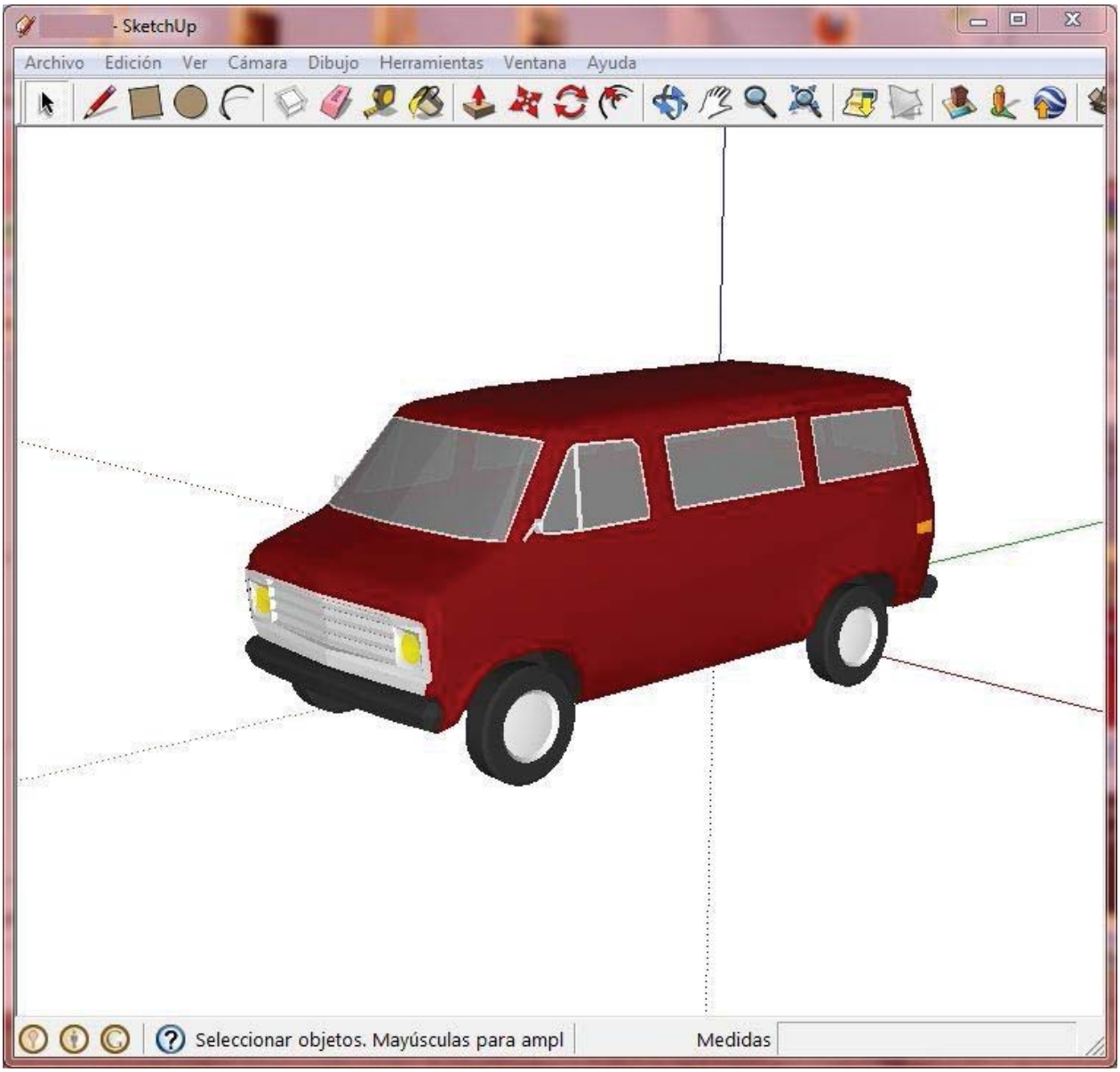

Ilustración 6-16 Modelo 3d del vehículo siendo diseñado en Sketchup

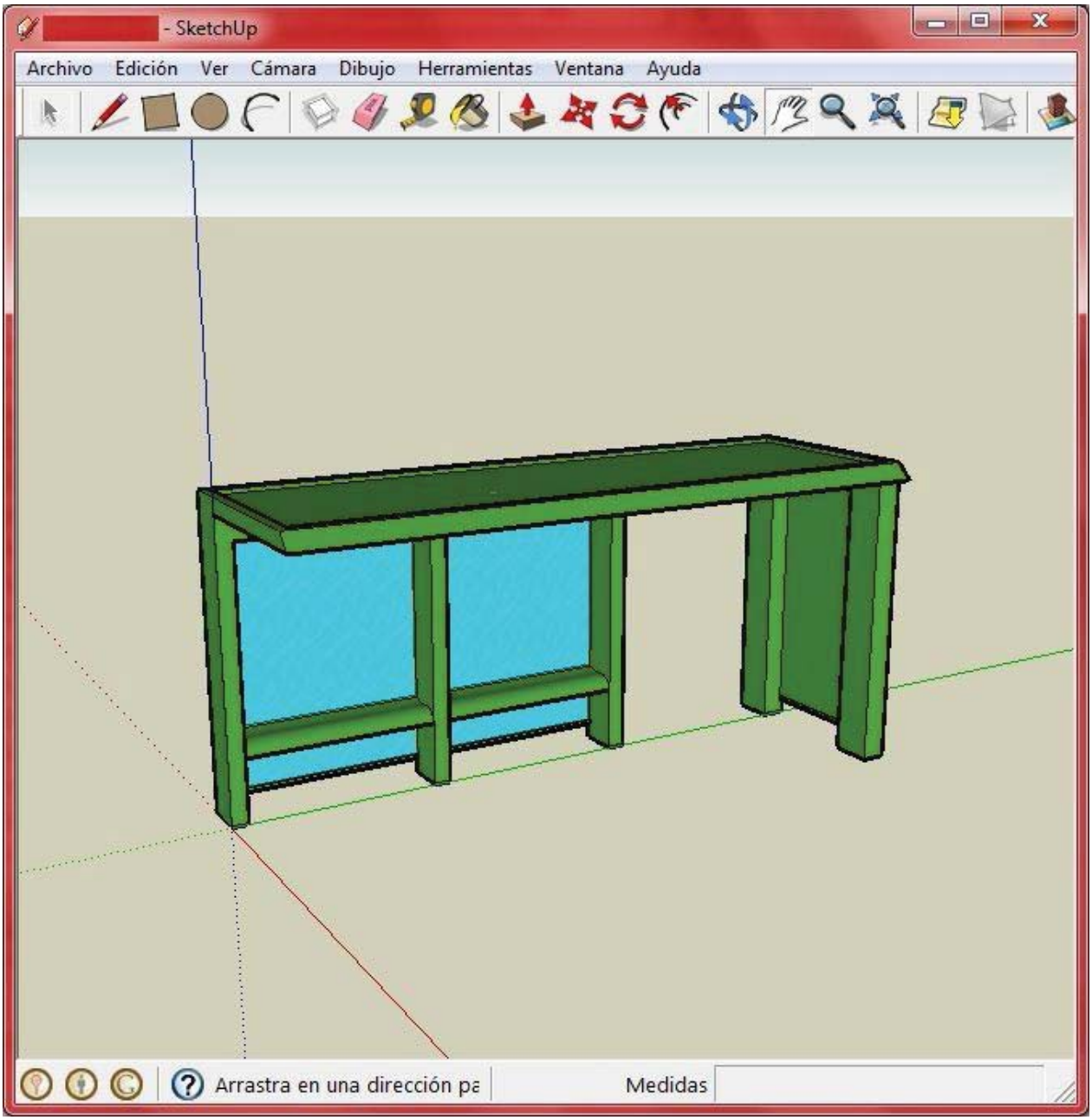

Ilustración 6-17 Modelo 3d del Paradero de Bus siendo diseñado en Sketchup

7 PRUEBAS AL SISTEMA

Las Pruebas en un sistema cumplen la tarea de verificar el Sistema para comprobar si cumple o no con los requisitos y especificaciones que menciona, además de verificar la interacción de los componentes y verificar la integración adecuada de estos. Se pueden desarrollar diversos tipos de pruebas en un mismo Sistema, algunas de estas pueden ser: funcionales, de usabilidad, de rendimiento, de prestaciones, de seguridad, instalación, entre otras. Las pruebas que se van a realizar están orientadas a la funcionalidad que presenta el Sistema, validando si el comportamiento que se efectúa cumple con las especificaciones que se plantean.

Existen distintos procesos y metodologías para las pruebas de software. La mayoría de ellos distinguen las etapas de planificación, diseño y ejecución, en el presente informe solo se registraran las primeras dos etapas. Durante la planificación de las pruebas se decide qué se probará y con qué profundidad. En la etapa de diseño de las pruebas funcionales, la especificación se analiza para derivar los casos de prueba y en la última etapa es donde se realiza la ejecución de los casos de pruebas diseñados comparando los resultados reales con los esperados, reportando los resultados.

7.1 Planificación de Pruebas

Con el fin de probar la integración del visualizador 3D en el sistema multiagente de transporte de pasajeros, se utilizará un escenario de pruebas fijo, determinado por 9 paraderos, 10 buses y 40 usuarios distribuidos aleatoriamente en los paraderos. El grafo formado por los paraderos se muestra en la Ilustración 7-1

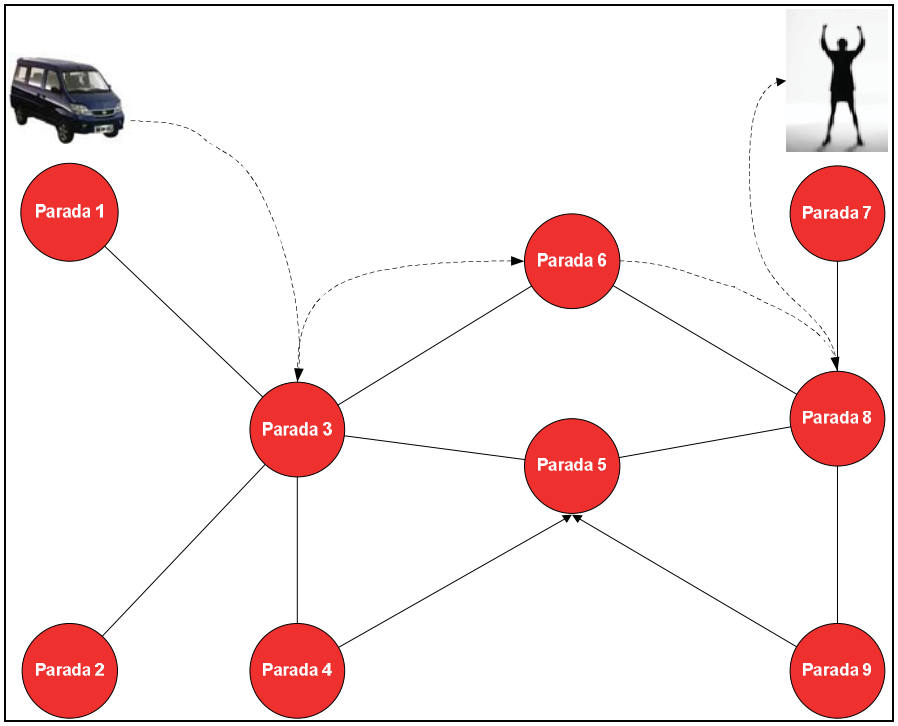

Ilustración 7-1 Grafo de Distribución de los Paradas

Como se menciona en capítulos anteriores, la forma en la cual el grafo se dibujará en el motor 3D está determinada por la información incluida en el archivo 3d.txt según el formato ya definido previamente.

Una vez definido el escenario, se deben determinar las pruebas a realizar, ellas son:

- Visualización de Eventos
- Cámara
- Carga de Modelos
- Sincronización con el Sistema Original

7.2 Diseño de Pruebas

En esta etapa se diseñan los escenarios de pruebas sobre las funcionalidades especificadas en la etapa anterior.

Visualización de Eventos

- Se lanzará la interfaz, y comenzará la simulación, visualizando el desplazamiento de los vehículos respecto a lo señalado por el Schedule.
    - Los eventos que ejecuten los vehículos visualmente deben ser los mismos que le son asignados en el simulador.
    - Para confirmar esto se deben comparar los eventos informados en el log, respecto a los realizados en la interfaz.

Cámara

- Existe un manejo libre del punto desde donde visualizar el sistema (cámara)
    - La cámara nunca debe pasar bajo el modelo de la base, ni detrás del cielo.
    - Se realizar un paneo completo de la cámara por toda la interfaz, comprobando que no pierda el punto de vista buscado.

Carga de Modelos

- Con el fin de visualizar los eventos, existe una gran cantidad de modelos 3d que son cargados en el sistema, es necesario que todos estos sean cargados correctamente.
    - Se debe verificar que todos los modelos sean los solicitados, y se encuentren cargados correctamente.
    - Verificar que las texturas se encuentren cargados correctamente.

Sincronización con el Sistema original

- Los eventos que se visualizan deben suceder al mismo tiempo que son generados
    - Si un evento comienza, debe hacerlo también su representación 3d, lo mismo en caso de termino.

8 CONCLUSIONES

Con la integración del motor gráfico 3D realizada en el sistema original, se ha logrado visualizar a los buses desplazándose de una parada a otra, según lo demandado por el sistema, además todo se encuentra trabajando de forma dinámica, adaptándose a cualquier distribución de pasajeras, paradas o buses, mientras que el archivo 3d_file.txt entregue la ubicación "visual" de los nodos. Las mayores dificultades en el trabajo realizado hasta ahora ha sido el nulo conocimiento previo respecto a cómo funciona y como se utiliza un motor de gráficos 3D, debido a esto el trabajo realizado ha sido lento pero arduo, más si contamos que luego de entender cómo funciona un motor 3D se procedió con el modelamiento de las diferentes estructuras 3D a utilizar lo cual también llevo a varias horas de investigación y aprendizaje.

Se crearon diferentes prototipos a fin de ver qué tal se comportaba el motor 3D, y cómo se realizaba la carga de los modelos en él. En un comienzo fue imposible lograr que los modelos se visualizaran correctamente, debido a que siempre poseían algún fallo en la carga de textura, luego de muchos intentos, se optó por utilizar otro formato de archivo para cargar los modelos (OBJ). Luego de esto la carga de los modelos se realizaba perfectamente.

Con respecto a la metodología de desarrollo se puede señalar que ofrece diversos diagramas que permiten ir visualizando la manera en que interactuarán los diversos elementos del sistema, además de que permiten modelar los aspectos más relevantes de un sistema multiagente. De este modo, la labor de modelado se torna bastante intuitiva al estar separada en fases bien definidas, cada una de las cuales constituye una especialización de la anterior. Por otra parte, la metodología permite la utilización de notación UML, lo que no hace necesario aprender una nueva forma de notación para su utilización, además, ésta hace uso de la notación básica de este lenguaje de manera de no complicar excesivamente los diagramas.

El objetivo principal de este proyecto fue generar las bases para generar una visualización de mucho mayor detalle gráfico, como pueden ser una mayor cantidad de animaciones, mejorar el entorno de visualización, etc. Durante el proyecto se investigaron e implementaron las herramientas básicas necesarias para lograr este objetivo, modificando los agentes claves que permitieran extraer la información relevante sobre los eventos, definiendo cuál era el mejor lugar a intervenir en el sistema para integrar el nuevo módulo de gráficos 3d, etc. Todo funcionando de forma transparente para el usuario y para el sistema mismo, sin generar problemas en la ejecución del flujo normal del proceso.

Con la realización de este trabajo, queda claro el potencial que tiene el representar de forma gráfica sistemas de este tipo. Tal vez se podría pensar para un trabajo futuro la generación de otros agentes que permitan extender aún más las funcionalidades básicas entregadas por los nuevos integrados. Queda pendiente integrar modelos que representen a los pasajeros, así como agregar animaciones a los modelos para dotar de mayor realismo al sistema.

REFERENCIAS

[**AgentLink, 2009].** *Software Products for Multi-Agent Systems.* s.l. : Technical report Europe's Network of Excellence for Agent-Based Computing. 2009

[Bellifemine et al., 1999] F. Bellifemine, A. Poggi y G. Rimassa, *JADE - A FIPA - Compleant Agent Framework.* s.l. : CSELT Internal Technical Report, 1999.

[Finin et al., 1993] T. Finin, D. McKay, R. Fritzson y R. McEntire, *KQML: An Information and Knowledge Exchange Protocol.* s.l. : . In Proc. of the International Conference on Building and Sharing of Very Large-Scale Knowledge Bases, 1993.

[FIPA, 2002]. *FIPA Request Interaction Protocol Specification.* s.l. : Standard N. SC00026H, 2002.

[Georgeff et al., 1998] M. Georgeff, B. Pell, M. Pollack, M. Tambey M. Wooldridge, *The Belief-Desire-Intention Model of Agency.* s.l. : In Proc. of ATAL '98: 5th International Workshop on intelligent Agents: Agent Theories, Architectures, and Languages, 1998.

[González et al., 2006] E. González, A. Hamilton, L. Moreno, G. Marichal y J. Mendez, *Diseño e Implementación de un Sistema Multiagente para la Identificación y Control de Procesos.* 2006.

[Hayes-Roth, 1995] B. Hayes-Roth, *An Architecture for Adaptative Intelligent Systems.* s.l. : Artificial Intelligence:Special Issue on Agents and Interactivity, pp. 329-365, Vol. 72, 1995.

[Hodjat y Amamiva, 2002]. B. Hodjat y M. Amamiva, *An Agent-Oriented Architecture for Natural Language Interfaces.* s.l. : In Proc. of SCI'02: 6th World Multiconference on Systemics, Cybernetics and Informatics Orlando , pp. 128-133, 2002.

[Krumpus, 2001]. M. Krumpus, *Overview of the Evo Artificial Life Framework.* s.l. : The Omicron Group, Technical Report, 2001.

[Odel et al., 2001] J. Odel, V. Parunak y B. Bauer, *Representing Agent Interaction Protocols in UML.* pp. 121-140, 2001.

[Russell y Norvig, 2003] J. Russell y P. Norvig, *Artificial Intelligence - A Moder Approach.* Upper Saddle River, New Jersey : Pearson Education, 2003.

[Weiss, 1999] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* s.l. : The MIT Press , 1999.

[Wooldridge y Jennings, 1995] Michael Wooldridge y R. Jennings, *Intelligent Agents : Theory and Practice.* s.l. : The Knowledge Engineering Review, Vol. 10, pp. 115-152, 1995.

[Donoso y Sandoval, 2009] Makarena Donoso P. y Daniel Sandoval U. , *Desarrollo de un Sistema para el Dynamic Dial-A-Ride Problem (D-DARP), utilizando tecnologia Agentes*,2009.

[Meyer, 2003] Wolfgang Meyer, Unión Internacional de Transporte Público (UITP).Disponible online en http://www.uitp.com/Events/madrid/mediaroom/Articles/seamless_mobility_es.htm. 2003.

[Cubillos, 2004] Claudio Cubillos F., MADARP : Multi-Agent Framework for Passenger Transportation Systems. Tesis Doctoral. Politecnico de Torino. 2004.

[Cordeau, 2003] Cordeau, J.-F., "A Branch-and-Cut Algorithm for the Dial-a-Ride Problem", Operations Research 54, 573-586, 2003. Canada Research Chair in Distribution Management, HEC Montréal 3000, Canada.

[Cordeau, Laporte, Ropke, 2006] Cordeau, J.-F., Laporte, G., Ropke, S., "Recent Models and Algorithms for One-to-One Pickup and Delivery Problems", The Vehicle Routing Problem, B. Golden, R. Raghavan, E. Wasil, eds., November 2006 (Publication GERAD-2006-68).

[Zerbst y Düvel,2004] Stefan Zerbst y Oliver Düvel 3D Game Engine Programming, 2004.

[Cossentino y Potts] M. Cossentino y C. Potts, PASSI: a Process for Specifying and Implementing Multi-Agent Systems Using UML

[Unreal] Unreal Developer Network, http://udn.epicgames.com/Main/WebHome.html

[Crystal Space] Crystal Space Official Website, http://www.crystalspace3d.org

[Java3d] Java 3d Official Website, http://java3d.java.net

[JMonkey] JMonkey Engine Version 2.0, http://jmonkeyengine.com

[Modelado3d] Conceptos Básicos de Modelado 3d , http://www.cristalab.com/blog/modelado-3d-fundamentos-basicos-c15531l/

[Sketchup] Sketchup, http://sketchup.google.com/

[3dStudioMax] Autodesk 3d Studio Max, http://usa.autodesk.com/3ds-max/

[Blender] Blender, http://www.blender.org/

A. ANEXOS

a. Estudio de Factibilidad

El desarrollo de cualquier proyecto informático siempre se enmarca dentro una serie de restricciones. Esto debido a que los recursos disponibles para llevar a cabo los proyectos siempre son limitados, implicando una gran cantidad de opciones que deben evaluarse para finalmente tomar las decisiones apropiadas que hagan un uso más eficiente de los recursos con los cuales se cuenta para la realización del proyecto. Este estudio juega un rol fundamental en cualquier proyecto informático que desee llevarse a cabo, ya que permite considerar variables como tiempo y dinero.

A continuación se mostrarán los resultados obtenidos en el análisis de la factibilidad técnica, económica, legal y operacional.

i. Factibilidad Técnica

Este proyecto en especial, pone un gran énfasis en si contamos con los recursos técnicos para llevarlos a cabos. Dentro de los recursos se cuentan la disponibilidad hardware y software, tecnología, recursos humanos y riesgos del desarrollo.

En esta primera etapa del proyecto se cuenta con la disponibilidad de los laboratorios de la Escuela de Informática de la Pontificia Universidad Católica de Valparaíso, a los cuales se permite el acceso seis días de las semana, a la investigación de las tecnologías a través de Internet, además del acceso a páginas con papers relevantes y que están disponibles en forma gratuita para su descarga

En cuanto a software, debido al hecho de que las plataformas de desarrollo de agentes actuales están desarrollas para ser trabajadas en Java, la elección de esta tecnología resulta implícita. Para la programación se utilizara un IDE el cual nos brindara un marco de trabajo más cómodo. El software elegido es Eclipse debido a que cuenta con un gran número de herramientas que no sólo facilitan la programación, sino que también provee un entorno en el cual la navegación entre clases no resulta compleja.

En vista de la experiencia existente se ha decidido trabajar usando la metodología de agentes PASSI, debido a que es conocida y que por lo mismo conlleva

menos riesgos que la elección de las otras. Se posee la experiencia en el desarrollo de MAS, lo cual aminora el riesgo debido a que se cuenta con experiencia acerca de estos sistemas y disminuye los riesgos que serán vistos en el capítulo respectivo a ellos.

ii. Estudio de Factibilidad

Debido al carácter del proyecto enfocado a la investigación, y no al uso empresarial, la tasa de retorno del capital no se considerará como un punto para tomar una decisión. Sin embargo existen otros puntos medibles, como son el costo del hardware necesario para desarrollar e implementar el sistema.

Tabla A-1 Costo Software

Herramientas	Costo
Motor Gráfico	0
Software de Modelado 3D	0
Eclipse	0
Jade	0
JAVA	0
Enterprise Architect	70.000
Total	**70.000**

Tabla A-2 Costo Hardware

Hardware	Costo
Equipo x1	300.000
Total	**300.000**

Si sumamos los costos de software y hardware, la suma mínima necesaria haciende a $370.000 aproximadamente, suma nada despreciable si tomásemos esto como un proyecto particular, sin embargo, la mayoría de las herramientas software nombradas poseen una licencia libre, y los gastos en las que son pagadas mas el hardware, podrían desaparecer utilizando los equipos y software proporcionados por los laboratorios de la universidad.

iii. Factibilidad Legal

Considerando que se intentara reducir al mínimo los costos involucrados en el proyecto, se potenciara la utilización de herramientas y software open-source, por lo cual, mientras se cumpla lo dicho en cada una de las licencias involucradas (apache, etc.) no existirían inconvenientes. Debido al hecho de que este proyecto se realiza en dependencias de la Escuela de Ingeniería en Informática de la Pontificia Universidad Católica de Valparaíso, se trabajará en lo equipos de este lugar los cuales cuentan con licencias.

iv. Factibilidad Operativa

Sobre las tecnologías y conocimientos necesarios para el desarrollo del sistema, refiriéndose con esto a Java, Jade, MAS y Gráficas 3D, además de la metodología utilizada, RUP, y el paradigma de orientación a agentes, existe conocimiento y manejo previo de la mayoría de estos puntos, por lo cual se puede asumir un buen desarrollo de ellos, y en caso de los que no son dominados, ya se encuentran en proceso de aprendizaje para minimizar las posibilidades de demora y error por estos.

b. Análisis de Riegos

Una estrategia proactiva ante los riesgos, nos permiten administrarlos de manera óptima, permitiéndonos mitigarlos y en casos extremos tener planes de contingencia con los cuales hacerles frente y que no afecten al proyecto de manera catastrófica, lo cual pueda incidir en el mayor gasto de recursos, ya sean de tipo de tiempo, humanos o monetarios.

Para esto entenderemos el riesgo como cualquier situación adversa, y a cada una de esas situaciones les asignaremos una probabilidad de ocurrencia y un plan de mitigación asociado. También se elaboraran planes de contingencia para aquellos riesgos que tengan un impacto alto en el proyecto.

Los riesgos pueden clasificarse en tres tipos:

- Riesgos del proyecto, amenazan la planificación del proyecto, afectan la calendarización o los recursos, por ello conllevan un retraso en los tiempos y un aumento en los costos.

- Riesgos de producto, afectan la calidad o desempeño del software que se está desarrollando, si los riesgos de este tipo se llegan a hacer realidad, la implementación puede volverse difícil o imposible.

- Riesgos del negocio, afectan a la organización que desarrolla o procura el software.

Para administrar los riesgos asociados al proyecto, se efectuarán las siguientes actividades:

- Identificación de los riesgos, identificar los riesgos de proyecto, negocio y producto.

- Análisis de riesgos, evaluación de la probabilidad y consecuencia de los riesgos.

- Planificación de riesgos, elaboración de planes para minimizar o evitar los efectos del riesgo.

- Monitoreo de riesgos, monitorear los riesgos durante todo el proyecto.

- Cada riesgo conlleva un plan de mitigación asociado el cual permite minimizar la ocurrencia de dicho riesgo, sin embargo, los planes de contingencia serán sólo realizados para aquellos riesgos

que tengan un alto grado de ocurrencia y que además tengan una incidencia en el proyecto que puedan de tal modo afectarlo en forma seria.

Tabla A-3 Identificación y análisis de riesgos

Descripción del Riesgo	**Tipo de Riesgo**	**Probabilidad**	**Impacto**	**Identificador**
Incumplimiento de los plazos establecidos.	Proyecto, Producto	70%	Catastrófico	1
Se retira o aparta algún integrante del equipo de trabajo.	Negocio	10%	Catastrófico	2
El o Los miembros del equipo no conocen o no dominan las tecnologías a utilizar para el diseño e implementación del proyecto.	Producto, Negocio	80%	Catastrófico	3
El dimensionamiento del proyecto no se mantiene dentro de los márgenes establecidos.	Proyecto	40%	Serio	4
No disponibilidad del software o hardware necesario para el desarrollo del proyecto.	Proyecto	20%	Serio	5
Los hitos de revisión de avance no son cumplidos.	Proyecto, Producto	60%	Serio	6
La herramienta de diseño es ineficiente.	Producto	30%	Tolerable	7
El tiempo dedicado al proyecto se ve disminuido por otras actividades.	Proyecto	60%	Serio	8
La Creación de los Modelos 3D es demasiado lenta.	Proyecto	50%	Tolerable	9
Implementación de Grafica 3D es sumamente dificultosa	Proyecto	40%	Serio	10

Tabla A-4 Planes de mitigación del proyecto

Identificador de Riesgo	Medidas para minimizar o mitigar el riesgo	Plan de contingencia asociado
1	Evaluación y seguimiento de las actividades a desarrollar en el proyecto.	1
2	Reuniones periódicas para evaluar y motivar a los integrantes del equipo.	-
3	Reuniones periódicas en las cuales se evaluaran el nivel de conocimientos de los integrantes y su nivel de aprendizaje.	2
4	Establecer los límites del sistema en forma temprana y clara.	3
5	Solicitud en los plazos establecidos de los equipos necesarios, y obtener las herramientas necesarias con tiempo de holgura.	-
6	Revisiones semanales de los grados de avance en las actividades a cumplir.	4
7	Realizar de manera temprana una revisión de las características de las herramientas, para decidir si continuar con la herramienta actual o elegir otra.	-
8	Planificar y dividir equitativamente los tiempos para la responder en cada una de las actividades.	5
9	Reducir al mínimo la cantidad de Modelos 3D requeridos	6
10	Búsqueda de Manuales o Sistemas 3D existentes para basarse en ellos.	7

Los planes de mitigación están asociados a todos los riesgos, que han sido identificados en el proyecto. De esta forma, cada riesgo observado cuenta con un plan para minimizarlo aunque la ocurrencia de dicho riesgo sea baja. Sin embargo, no todos los riesgos cuentan con planes de contingencia, esto debido a que los planes de contingencia tienen un costo asociado, el cual incide en el proyecto, es por ello que estos planes se generan solamente para aquellos casos en los cuales la ocurrencia del riesgo está entre el rango medio y alto y además tenga una incidencia crítica o catastrófica en el proyecto.

Tabla A-5 Planes de contingencia del proyecto

Plan de Contingencia	Descripción del plan de contingencia.
1	Ajustar nuevamente los plazos, para cumplir con las fechas fijadas.
2	Comenzar con horarios establecidos en los cuales se realizarán cursos y charlas sobre las tecnologías a utilizar.
3	Volver a fijar los límites del sistema y dejar en forma escrita una constancia que los establezca.
4	Fijar horarios de trabajo fijos, y penalizar el incumplimiento de estos horarios.
5	Establecer horas fijas a cada actividad y velar por el cumplimiento de ellas.
6	Simplificar el diseño de los Modelos 3D requeridos.
7	Reemplazar el ambiente 3D por uno 2D

B. Diagramas

a. TripGen Agent

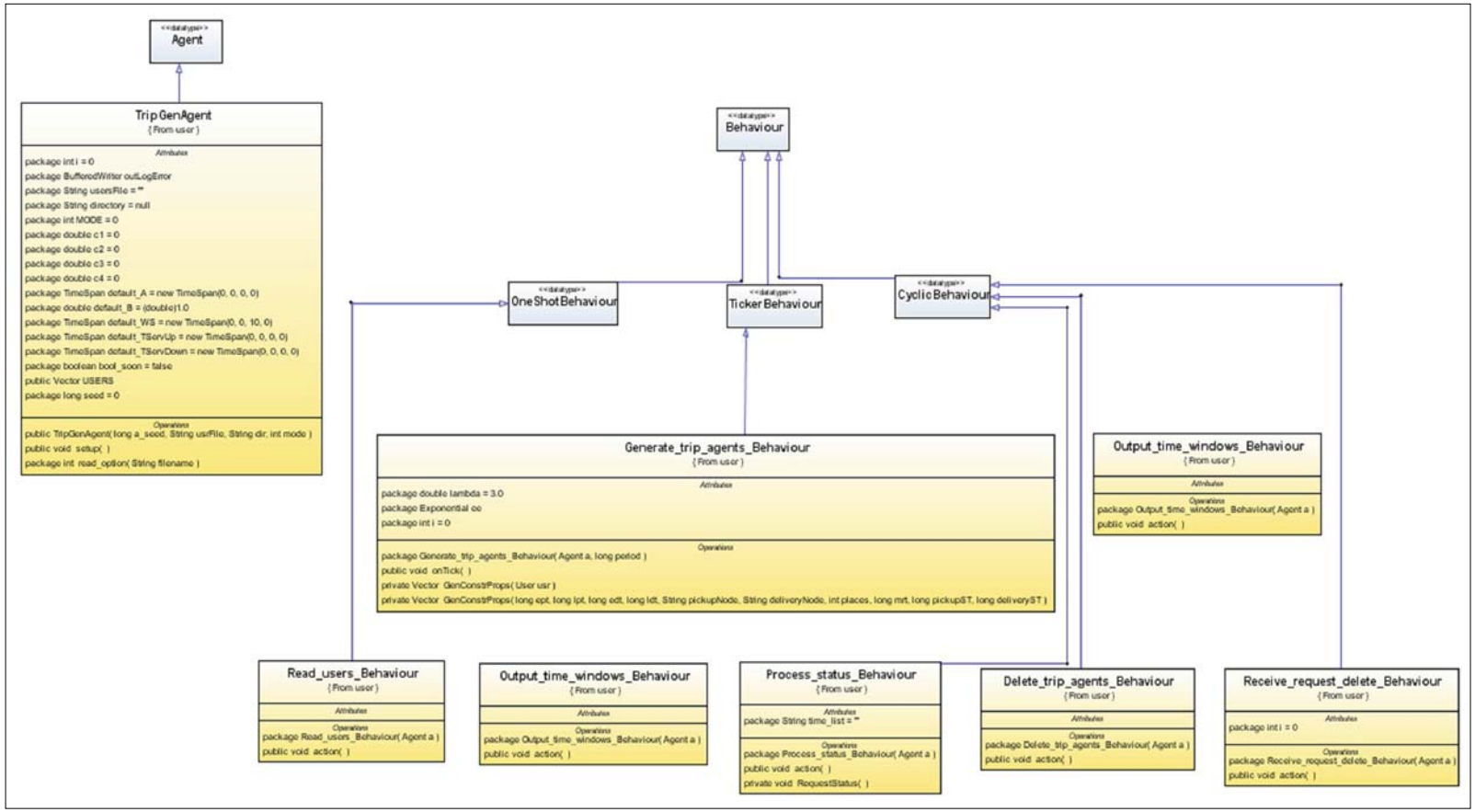

Ilustración B-A-1 Diagrama de Estructura TripGen Agent

b. SchedGen Agent

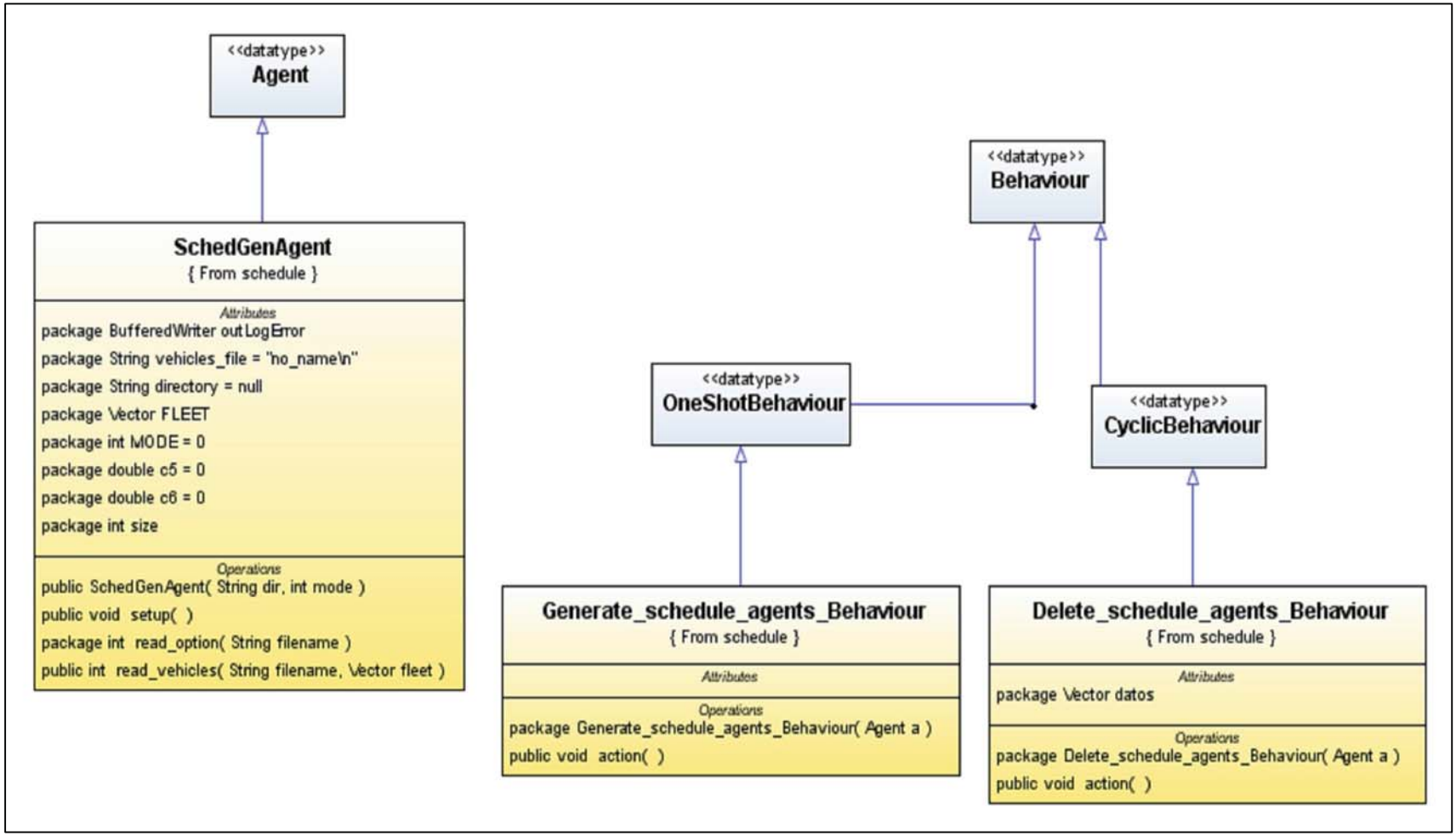

Ilustración B-A-2 Diagrama de Estructura SchedGen Agent