Universidad de las Ciencias Informáticas

Facultad 5

MÓDULO DE COOPERACIÓN ENTRE AGENTES VIRTUALES PARA EL SISTEMA DE PERCEPCIÓN

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autora: Saily Hurtado Gracia

Tutora: Ing. Yenifer del Valle Guevara

Co-Tutor: Ing. Alexey Broche Medina

La Habana, Junio del 2011
"Año 53 de la Revolución"

"El aspecto fundamental en el cual la juventud debe señalar el camino es precisamente en el aspecto de ser vanguardia en cada uno de los trabajos que le compete."

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste, firmamos la presente a los ____ días del mes de _____ del año_____.

_____

Autora: Saily Hurtado Gracia

_____ _____

Tutora: Ing. Yenifer del Valle Guevara **Co-Tutor:** Ing. Alexey Broche Medina

DATOS DE CONTACTO

Generales de la Tutora:

Nombre y apellidos: Ing. Yenifer del Valle Guevara

Especialidad: Ingeniería en Ciencias Informáticas

Años de experiencias: 4 años

Correo electrónico: ydelvalle@uci.cu

Teléfono de contacto: 766-7474

Generales del Co-Tutor:

Nombre y apellidos: Ing. Alexey Broche Medina

Especialidad: Ingeniería en Ciencias Informáticas

Años de experiencias: 2 años

Correo electrónico: abroche@uci.cu

Teléfono de contacto: 837 2261

DEDICATORIA

Va dedicada a:

A mi mamá por ser la mejor madre y persona del mundo, por guiarme y apoyarme en todo momento, por ser tan especial.

A mi papá por ser mi paradigma a seguir, por ser increíblemente especial.

A los dos: por ser los motores propulsores de esta obra…aquí está el resultado de tanto sacrificio y esfuerzo… nada ha sido en vano.

Los ama

Su nena.

AGRADECIMIENTOS

A mis padres por haberme dado su apoyo en todo momento, por confiar en mí y demostrarme que sí se puede, por ser las personas que más significan para mí, por ser los mejores padres que un hijo pueda tener.

A mi Papo por ser el mejor hermano-padre que una pequeña pueda tener, gracias por ser ejemplo clave en mi vida, gracias por tu dedicación y cariño. Así como mismo fuiste tú para mí, espero ser yo para mi sobrino.

A mi familia completa por formar parte de mi día a día.

A mis tutores por dedicarme tiempo a pesar de ser personas muy ocupadas, por transmitirme sus conocimientos y confiar en mí.

A mi gran amiga Yaneiry por su apoyo y confianza en todo este tiempo, a su familia por quererme como una hija más.

A todas mis amistades, de la universidad y del pre, por enseñarme el valor de un amigo y por apoyarme en todo momento: Cecy, Las Sandras, Yahire, Brenda, Leticia, Karla y Osmany.

A todas mis compañeras y compañeros: Dayana, Yeni, Lora, Mailén, Adis, Anna, Yuneisis, Susana, Coca, Adriel, Juli, Adrián Carlos, Yadira (la profe), Arián, Osvaldo, Jesús y su familia.

A mi pequeño equipo de trabajo: Ray y el Chino, por ayudarme.

A mi tribunal por haberme guiado y corregido en todos los cortes.

A mi oponente por ser exigente y tan buena persona.

A todas aquellas personas que han contribuido a mi formación como profesional.

A Elena por dejarme ser parte de su vida y quererme como si fuera una hija, gracias por estar pendiente de mí.

A Fidel y a la Revolución por haberme dado la oportunidad de estudiar en esta universidad de excelencia.

Hoy soy lo que soy gracias a la ayuda de todos.

Gracias de corazón:

Say.

RESUMEN

En la actualidad informática, muchos han sido los avances que han tenido lugar en las diferentes materias y ramas de la misma, no quedando exenta la Inteligencia Artificial, y dentro de ella uno de sus principales ejes: los Sistemas Multi-Agentes. Los mismos se han encargado de estudiar el modelo y comportamiento de varios agentes en un entorno, y ofrecen un sinnúmero de prestaciones de manera intuitiva y segura en aplicaciones que se encuentran dentro de las siguientes ramas: educación, salud y entretenimiento.

Este trabajo propone desarrollar un módulo de cooperación entre agentes virtuales a partir del estudio realizado sobre algunas de las diferentes técnicas que existen para lograr que los agentes intercambien información. Dándole respuesta a los objetivos y necesidades que éstos puedan presentar en un determinado entorno virtual.

En este documento se exponen los artefactos ingenieriles, de los diferentes flujos de trabajo, obtenidos al aplicar la metodología de desarrollo de software seleccionada. Así como el diseño e implementación de un conjunto de clases que permiten el intercambio de información entre dichos agentes.

Se obtuvo un módulo que brinda las funcionalidades necesarias para crear grupos de agentes, actualizarlos, eliminarlos, registrar agentes a un grupo y eliminarlos y registrar los datos de interés a cada grupo creado; para su acople al Sistema de Percepción.

Los resultados expuestos en esta investigación demuestran que con el uso del módulo, se facilita la cooperación entre agentes y se reducen los cálculos de los algoritmos de percepción que antes existían en el proyecto Entrenadores Aduaneros, desarrollado en la Facultad 5 de la Universidad de las Ciencias Informáticas.

Palabras clave: Agentes, Cooperación, Entorno virtual, Percepción.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICAS

INTRODUCCIÓN

La Inteligencia Artificial (IA) es la disciplina que se encarga de construir procesos que, al ser ejecutados sobre una arquitectura física, producen acciones o resultados que maximizan una medida de rendimiento determinada, basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura; puede ser considerada también, como “el arte de construir máquinas capaces de hacer cosas que requieran de inteligencia en caso de que fuesen hechas por seres humanos.” [12]

Dentro de sus ejes fundamentales se encuentran los Sistemas Multi-Agentes (SMA), los cuales se han encargado de estudiar el modelo y comportamiento de varios agentes que cooperan entre sí para desarrollar un trabajo, y se comunican por medio de mecanismos basados en el envío y recepción de mensajes. [8]

Algunos autores definen a los agentes como los nodos inteligentes por los cuales están compuestos los SMA [1]. Aunque no existe una definición formal y precisa de lo que es un agente, ya que muchos expertos lo asocian a su problema específico, se sabe que éstos pueden comunicarse con otros agentes dentro o fuera de su ambiente para obtener información y lograr, de forma más eficiente, el cumplimiento de sus objetivos; y que actúan sobre el medio en el que se desenvuelven a través de ejecutores. Estos agentes, realizan una serie de operaciones destinadas a cumplir un objetivo, ya sea por iniciativa propia o porque dada una situación se lo requiera. Pueden, además, realizar procesos de manera continua, los cuales les ayudarán a conocer cómo hacer sus tareas.

Debido a las múltiples funcionalidades que brinda el trabajo con agentes, en la Universidad de las Ciencias Informáticas (UCI), actualmente en el Centro de Informática Industrial (CEDIN), se está trabajando en la elaboración de un Sistema de Percepción (SP), que será el encargado de brindarle, a dichos agentes, todos los datos del medio que los rodea en caso que éstos los necesiten.

Mediante la obtención de estos datos, los agentes pueden tomar las decisiones necesarias para poder resolver un problema determinado; para ello es preciso que puedan intercambiar información con otros agentes del entorno, pues “la resolución de los problemas que tengan estos agentes, es sólo válida si los agentes integrados al sistema en cuestión, son capaces de comunicarse entre sí para darle una solución a los mismos”. [13]

Este SP no posee la funcionalidad de permitir que los agentes intercambien información entre sí, que es a lo que se llamará *cooperación o comunicación* en el transcurso de esta investigación; pues al tener la misma almacenada no se puede reutilizar en el momento que algún agente lo necesite, trayendo redundancias en los cálculos de los algoritmos de percepción que existen actualmente; implicando a su vez, la inexistencia de retroalimentación entre los agentes, al necesitar determinado(s) dato(s) para resolver algún problema que ya haya sido resuelto por otro agente.

Se resume, por lo anteriormente explicado, que el **problema** existente en el SP es el siguiente:

- No existe intercambio de información.
- Existe redundancia en los cálculos de los algoritmos de percepción.

Por la situación problémica anteriormente explicada, se define como **problema científico:** ¿Cómo gestionar la cooperación continua entre agentes del Sistema de Percepción?

Teniendo como **objeto de estudio** de la investigación: Los Sistemas Multi-Agentes en el proceso de cooperación.

Y para darle solución al problema de investigación se formula como **objetivo general:** Desarrollar el módulo de cooperación entre agentes virtuales para el Sistema de Percepción.

Lo cual delimita como **campo de acción:** La cooperación entre agentes de un determinado entorno virtual.

Luego, para darle cumplimiento al objetivo general se plantean como **tareas investigativas**:

1. Elaboración del marco teórico de la investigación a partir del estado del arte existente actualmente sobre el tema de investigación.
2. Selección de la metodología de desarrollo de software, lenguaje y herramientas a utilizar.
3. Selección de la (s) técnica (s) a utilizar en el módulo de comunicación.
4. Generación de los artefactos ingenieriles según los flujos de trabajo que proponga la metodología seleccionada.
5. Implementación del módulo a partir de los requisitos planteados.
6. Validación de las funcionalidades del software mediante el análisis de los resultados obtenidos.

Idea a defender: La inclusión del módulo de cooperación permitirá que los agentes del entorno tomen los datos necesarios de él para resolver sus necesidades; y que compartan información con otros agentes, logrando eliminar la redundancia en los cálculos de los algoritmos de percepción existentes.

Para el desarrollo de la investigación se utilizaron diferentes métodos científicos, en los niveles teórico y empírico.

Nivel teórico:

- **Histórico – Lógico:** Permite estudiar la evolución y desarrollo de las técnicas que existen para lograr que los agentes intercambien información entre sí y conocer el estado actual de las mismas.
- **Analítico – Sintético:** Permite concretar y resumir el conocimiento reflejado en los materiales consultados sobre el tema de SMA para utilizarlo en el desarrollo de esta investigación.

Nivel empírico:

- **La consulta de fuentes de información:** Permite realizar un estudio actual de las distintas bibliografías que existen del tema tratado.
- **Observación científica:** Permite detectar el comportamiento desempeñado por los agentes controlados por el SP para identificar errores que afecten su accionar con el entorno que los rodea.

El contenido de este documento está estructurado en tres capítulos, así como las correspondientes secciones de las recomendaciones, bibliografía, el glosario de términos, glosario de abreviaturas y los anexos; organizados de la siguiente forma:

Capítulo 1: Fundamentación teórica

Se exponen los principales conceptos relacionados con los SMA y con el intercambio de información entre los agentes, partiendo del estudio del estado del arte referente al tema tratado; así como la descripción de la arquitectura que sustenta la base de formación del SP.

Capítulo 2: Propuesta y descripción de la solución

Se ofrece una descripción de las características que presentará el sistema como solución al problema científico planteado. Serán definidas, también, la metodología de desarrollo de software, el lenguaje de modelado, el lenguaje de programación y la herramienta case; que permitirán dirigir y guiar el proceso de desarrollo para poder llegar a los resultados esperados.

Se realiza además, la modelación del negocio y la captura de los requisitos funcionales y requisitos no funcionales con la generación de sus respectivos artefactos.

Capítulo 3: Diseño, implementación y resultados

Este capítulo incluye la generación de los artefactos, según la metodología seleccionada, para los flujos de trabajo diseño e implementación; así como el análisis de los resultados obtenidos al desarrollar la aplicación y una pequeña descripción de la misma.

CAPÍTULO 1

FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se realiza una evaluación del estado del arte referente al estudio de los Sistemas Multi-Agentes y de algunas de las técnicas que existen para poder lograr el intercambio de información entre los agentes de un determinado entorno virtual y sus características, así como la descripción de la arquitectura que presenta el SP al cual se le añadirá el módulo que se desarrollará para darle cumplimiento a los objetivos planteados.

1.1 Sistemas Multi-Agentes

Los SMA son sistemas compuestos por nodos o elementos inteligentes de IA donde la conducta combinada de dichos elementos, produce un resultado en conjunto inteligente, comunicándose por medio de mecanismos basados en el envío y recepción de mensajes. [8]

1.1.1 Agentes

Los nodos o elementos inteligentes son los llamados agentes. Los cuales son la base de la construcción de los SMA y son vistos como entidades inteligentes que pueden percibir su ambiente a través de sensores. Éstos son capaces de evaluar tales percepciones y tomar decisiones por medio de mecanismos de razonamiento sencillo o complejo. (Figura 1.1)

Muchos expertos han dado su criterio en relación con dicho término y de acuerdo a la situación en que se encuentran; algunos de los cuales se mencionan a continuación:

- “Un agente es una entidad que percibe y actúa sobre un entorno”. [2]
- “Los agentes son sistemas computacionales que habitan en entornos dinámicos complejos, perciben y actúan de forma autónoma en ese entorno, realizando un conjunto de tareas y cumpliendo objetivos para los cuales fueron diseñados”. [2]

- "Un agente es un sistema situado en alguna parte de un entorno que percibe dicho entorno y actúa en él en beneficio de su propia agenda, el efecto de su actuación se nota en el entorno". [8]

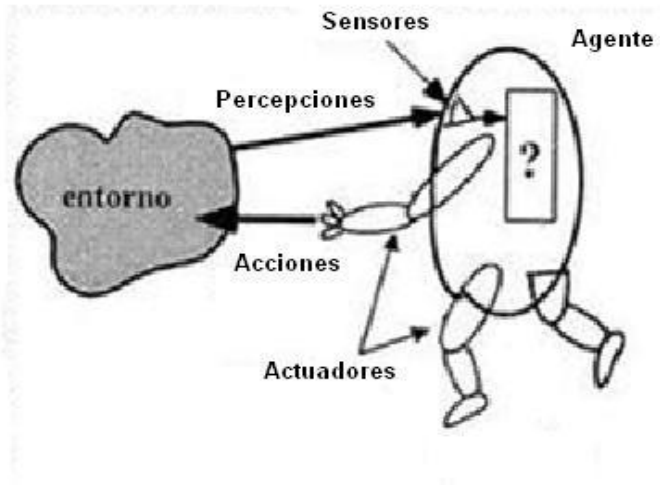


Figura 1.1 Representación de un agente

Pero todos concuerdan en que presentan determinadas características que son importantes destacar.

1.1.2 Características de un agente

Las características de los agentes vienen dadas por una serie de calificativos, los cuales denotan ciertas propiedades a cumplir por los mismos:

- **Reactivo:** Los agentes deben ser capaces de percibir el ambiente que los rodea y responder a tiempo a los cambios en él, a través de sus acciones.

- **Pro-activos:** Deben exhibir un comportamiento orientado por sus metas, tomando la iniciativa para satisfacer sus objetivos.

- **Social:** Deben ser capaces de interactuar con otros agentes con miras a lograr la satisfacción de sus objetivos.

Es por estas razones que en el desarrollo de esta investigación, se considerará a un agente como: *una entidad inteligente, capaz de percibir el entorno que lo rodea por medio de mecanismos basados en el envío y recepción de mensajes; interactuar con él y con otros agentes y tomar las decisiones que sean necesarias para darle solución a los problemas u objetivos que se le puedan presentar.*

Estos agentes, para poder llevar a cabo el cumplimiento de sus tareas y objetivos, necesitan poder intercambiar información entre sí, dándole a este proceso el nombre de comunicación virtual. Sin la cual no llegarían a resolver los problemas y objetivos que puedan tener definidos en el medio que los rodea.

1.1.3 Técnicas de comunicación entre agentes

En la actualidad son variadas las formas de lograr una comunicación entre varias personas gracias a la utilización de los medios informáticos, tales como los ordenadores. Al intercambio de información mediante esta vía es que se le llamará *comunicación virtual*, pues se emite una gran cantidad de información mediante el uso de la red sin la participación directa de seres humanos.

Hoy día existen diferentes formas de comunicación virtual, dentro de ellas se pueden mencionar:

- Correo electrónico.
- Mensajería instantánea.
- Videoconferencia.
- Chats.
- Foros.
- Blogs.

La comunicación virtual, dentro de la IA, permite que los elementos que se encuentran dentro de un entorno determinado puedan comunicarse entre sí sin la necesidad de que intervenga algún ser humano. [1]

Esta comunicación es definida, en este contexto, como la transmisión de cierto contenido informativo a un receptor mediante el uso de una señal. Forma base de la coordinación entre los agentes, pues puede darse el caso de que surja algún conflicto entre dos o más agentes, y es por eso que se hace necesaria la comunicación entre los mismos.

Dentro de las técnicas de comunicación que existen y permiten que dos o más agentes se comuniquen están: los Sistemas de Pizarra (*BlackBoard*), los Sistemas de mensaje/diálogo, los Sistemas Basados en Disparadores (*Trigger*), las Máquinas de Estado Finitos (MEF), entre otras (ver Anexo 1).

1.1.3.1 Sistemas de Pizarra (*BlackBoard*)

Es uno de los sistemas más usados en la IA. La pizarra es considerada una estructura de datos que es utilizada como mecanismo general de comunicación entre las múltiples fuentes de conocimientos, siendo gestionada y arbitrada por un controlador.

Esta pizarra es manejada como un área de trabajo común, donde los agentes pueden compartir cualquier tipo de información con los demás agentes del entorno. Es un dispositivo para compartir información con múltiples lectores y escritores (Figura 1.2), pues permite que cada agente trabaje en su problema, escoja de la misma lo que necesite para resolverlo y lo publique una vez encontrada su solución para que sirva de retroalimentación a los demás agentes del entorno.

En este tipo de pizarras no existen áreas privadas, los agentes pueden escribir la información que deseen y acceder a la misma cuando estimen necesario, trayendo como consecuencia que pueda ocurrir una sobrecarga a la pizarra según la cantidad de agentes que intenten acceder a la misma vez.

Para lograr la solución a este problema, se haría necesario una nueva pizarra donde cada uno de los agentes del entorno tendría asignado su propio espacio para publicar información en dicha pizarra y contarían de permisos totales de acceder, pero sin modificar la información que los demás hayan publicado.

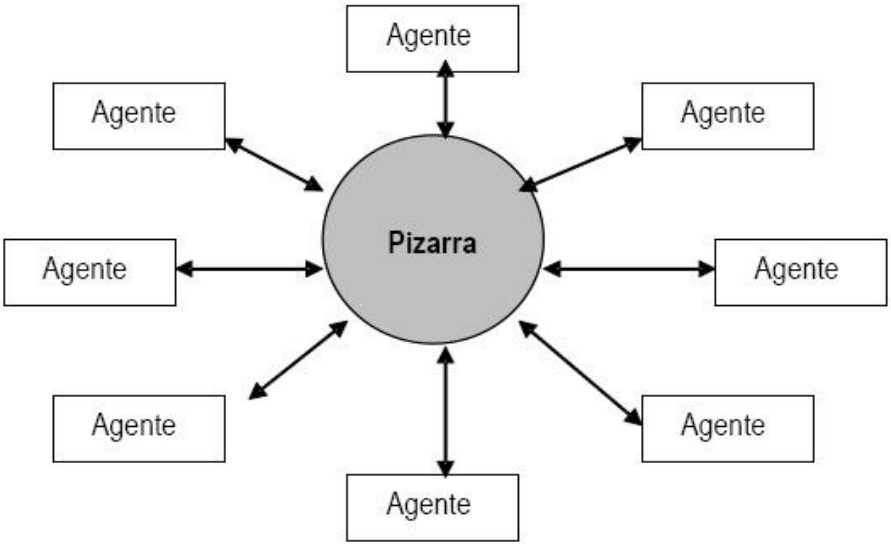


Figura 1.2 Estructura de un sistema de pizarra [12]

Este sistema tiene como ventajas:

- Es muy útil cuando el problema a resolver es extremadamente grande o no se tiene un sistema completo del sistema a resolver.
- No existe garantía, en la mayoría de los casos, de que se alcanzará una solución, pero si ésta se encuentra; es la mejor de todas.
- Cada agente tiene sus propios objetivos, aunque desconoce los objetivos de los demás y la solución del problema, pero puede llegar a obtener la información que necesita por el intercambio que pueda realizar con sus demás compañeros.

Presenta dentro de sus desventajas:

- Es una técnica ineficiente, puesto que no existe una cota exacta respecto al tiempo de cómputo necesario para resolver el problema, pero se puede adaptar a problemas en los que se conocen bien sus características. Así que esta desventaja se puede convertir en una gran ventaja en dependencia de las características del sistema en cuestión.
- Es difícil obtener una traza de los pasos que llevaron a la solución, es decir, no ofrece explicaciones a la hora de llevar a cabo la realización del proceso. [13]

1.1.3.2 Sistemas de mensaje/diálogo

Este sistema empieza a funcionar cuando un agente, conocido como emisor, transfiere un mensaje específico a otro agente, denominado receptor. (Figura 1.3)

Al contrario de los Sistemas de Pizarra, aquí los mensajes son intercambiados directamente entre dos agentes, donde los demás agentes son incapaces de leer dicho mensaje si no va específicamente dirigido a ellos, siendo ésta su principal ventaja o desventaja desde el punto de vista que sea analizada la situación.

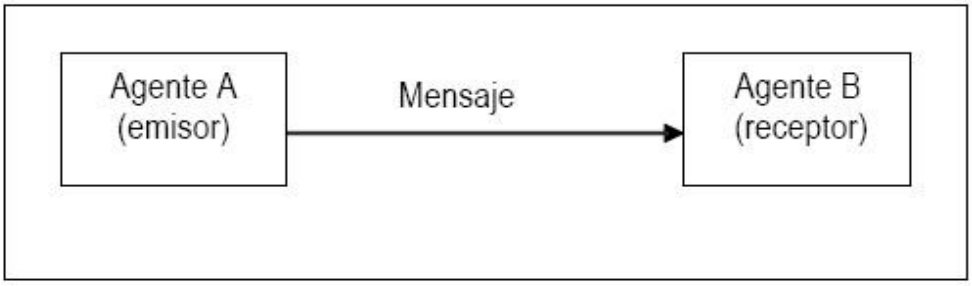


Figura 1.3 Principio de la transmisión de un mensaje [12]

1.1.3.3 Sistemas Basados en Disparadores (*Trigger*)

Estos sistemas constan de dos efectos esenciales en el mundo virtual:

- ✓ Realizar un seguimiento de los acontecimientos que los agentes puedan responder en el mundo.
- ✓ Minimizar la cantidad de agentes en proceso para responder a dichos eventos. [13]

Un disparador o *trigger* puede ser cualquier estímulo que requiera de un agente para responder. En un entorno determinado, los factores desencadenados podrían ser cualquier cosa audible o visible, que afecta el comportamiento de los agentes, tales como: disparos de armas de fuego, explosiones, enemigos cercanos, ruidos de las puertas al cerrarse o los cadáveres.

Los factores desencadenantes también pueden emanar de objetos inanimados que los agentes necesitan conocer, como palancas y paneles de control, para ello es necesario definirle a cada agente qué tipo de factores desencadenantes son de interés para ellos.

Un disparador se define por un conjunto de banderas enumeradas del mismo tipo, y un conjunto de variables que describen sus parámetros (Figura 1.4).

La combinación de banderas y de bits, permite a un agente especificar cuál es la acción que va a realizar. Un agente puede alternar y desactivar banderas e ignorar temporalmente determinados tipos de activación.

```
enum TipoDisparador
{
        K_Ninguno  = 0,
        K_Explosión = (1 << 0),
        K_EnemigoCerca = (1 << 1),
        K_Disparo = (1 << 2),
        ...
};
```

Figura 1.4 Ejemplo de los estados de las banderas [12]

El beneficio de un sistema centralizado de activación, como también se les conoce, es la verificación de la prioridad antes de entregar la información al agente, de esta forma; el agente sólo responde a los procesos de mayor prioridad dentro del entorno.

En un sistema centralizado los factores desencadenantes son registrados cuando se producen los acontecimientos. Para cada agente, el sistema utiliza una serie de pruebas para determinar si el agente está interesado en cualquier factor desencadenante existente.

Estos factores son ordenados por prioridad, a fin de que el agente pueda responder a la acción más importante en cualquier instante. Ejemplo: en un juego, si un enemigo está de pie delante de un agente, éste no se preocupa por los sonidos que aparezcan en la distancia, por lo que su prioridad número uno pasa a ser el enemigo que está delante de él.

Los factores desencadenantes pueden ser utilizados para alertar a los agentes de los sucesos del entorno, como pueden ser: disparos de armas de fuego y explosiones que ocurran. Un agente herido puede usar una forma dinámica de colocar una señal de activación cuando necesita ayuda de sus aliados. Un agente puede, incluso, detectar con lo que el jugador ha interactuado a través de objetos interactivos en el mundo virtual, tales como puertas y otras acciones que se disparan en el momento que son llamadas.

Este sistema presenta como ventajas:

- Verifica la prioridad antes de entregar la información al agente.

- Pueden ser generados a partir de una variedad de fuentes, incluyendo códigos de juego, scripts, comandos de consola, animación y fotogramas clave.
- Permiten implementar programas basados en paradigma lógico (ejemplo: sistemas expertos).
- Son sistemas muy eficientes y rápidos, por lo que se garantiza encontrar siempre una solución.

Desventajas:

- No pueden ser llamados directamente.
- Dada su complejidad son difíciles, pero no imposibles de programar.

1.1.3.4 Máquinas de Estados Finitos

Es una de las técnicas más relevantes para el desarrollo de los juegos de IA. Son simples, eficientes, fáciles de implementar y lo suficientemente poderosas para manejar una amplia variedad de situaciones.

Una Máquina de Estado Finito (MEF) o también conocidas como Autómatas de Estados Finitos, son modelos de comportamiento de un sistema, con un número limitado de estados o condiciones predefinidas y de transiciones entre dichos estados. Además, pueden definir un conjunto de condiciones que determinan cuándo el estado debe cambiar.

Las MEF son las técnicas más usadas en la implementación de los juegos en la IA, permitiendo una mejor comunicación entre los agentes que interactúan en dichos juegos.

Esta técnica de cooperación representa una descripción no lineal de cómo los objetos pueden cambiar de estado con el tiempo, posiblemente en respuesta a los eventos en su entorno. Además, se prestan para ser esbozadas con rapidez durante el diseño y prototipo, y mejor aún, pueden ser implementadas de manera fácil y eficientemente. [13]

La Figura 1.5 muestra un comportamiento basado en MEF donde los estados describen cómo los agentes van a actuar, y las transiciones entre estados representan las decisiones que tomarán éstos en el próximo paso.

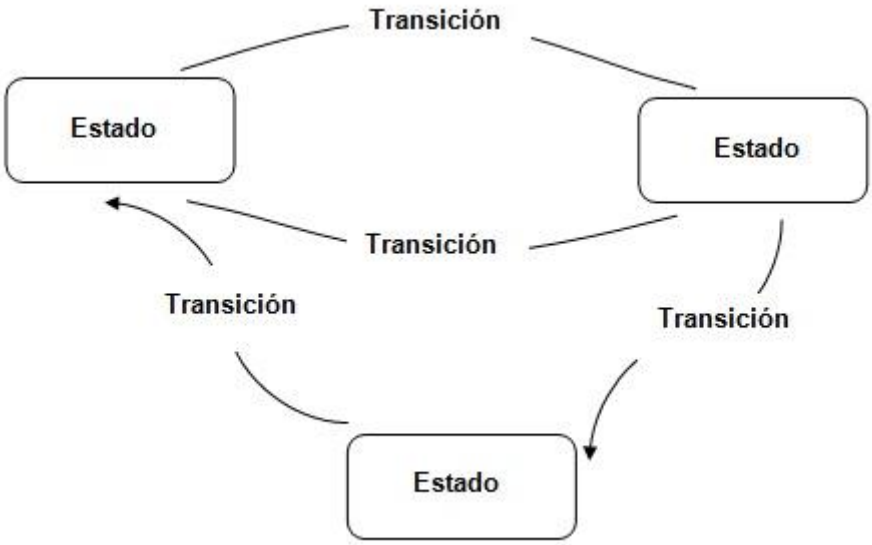


Figura 1.5 Ejemplo de una MEF como diagrama

Ventajas:

- Su simplicidad hace fácil, para los desarrolladores sin experiencia, realizar la implementación con poco o nada de conocimiento extra (fácil entrada).
- Predictibilidad, dado un grupo de entradas y un estado actual conocido, puede predecirse la transición de estados, facilitando la tarea de verificación.
- Dada su simplicidad, son rápidos de diseñar, rápidos de implementar y rápidos de ejecutar.
- Son relativamente flexibles.
- Pueden manejar una amplia variedad de situaciones.
- Presentan varios modos de implementarse partiendo de su topología.
- La transferencia desde una representación abstracta del conocimiento a una implementación es fácil.
- Bajo uso del procesador; apropiado para dominios donde el tiempo de ejecución está compartido entre varios módulos o subsistemas. Sólo el código del estado actual ha de ser ejecutado, además de requerir de un poco de lógica para determinar el estado actual.

- Es fácil determinar si se puede llegar o no a un estado, en las representaciones abstractas, resulta obvio si se puede o no llegar a un estado desde otro, y qué requerimientos existen para hacerlo.

Desventajas:

- La naturaleza predecible de las MEF puede no resultar conveniente en algunos dominios, como los juegos por ordenador.
- Si se implementa un sistema grande, usando esta técnica, puede ser difícil de administrar y mantener sin un buen diseño.
- No es apropiado para todos los dominios de problema, sólo debe ser usado cuando el comportamiento de un sistema puede ser descompuesto en estados separados con condiciones bien definidas para las transiciones, lo que significa que todos los estados, transiciones y condiciones deben ser conocidos y estar bien definidos con anterioridad.
- Las condiciones para las transiciones entre estados son rígidas, significando que están fijadas.

1.2 Sistema de Percepción

Para poder lograr que los agentes se comporten de manera real, es necesario dotarlos de la capacidad de percibir el entorno que los rodea de manera que puedan responder a los cambios que el mismo les ofrece.

Últimamente se han desarrollado múltiples sistemas de percepción ([16], [21], [22], [28]) con diferentes finalidades, en dependencia de la necesidad a satisfacer por parte de los usuarios o clientes finales; que presentan y documentan múltiples usos como: detección, monitorización y modelado de fuegos forestales, protección del medio ambiente; simulación de la percepción sensitiva del hombre, haciendo énfasis en los datos medioambientales, visión, auditivos y en la memoria de los agentes, de un determinado entorno virtual. Pero no dejan reflejado cómo es que tratan, o se haría, la cooperación o intercambio de información entre los agentes del entorno.

Es por ello que se hace necesario trabajar con una arquitectura de un sistema de percepción, que permita añadir a sus características, o reutilizar de ella, las bases necesarias para lograr que los agentes intercambien información entre ellos y con el entorno.

De ahí que surja un SP [11] que será el encargado de otorgarles a los agentes las habilidades de observar, escuchar y conocer las características topológicas y medioambientales del entorno que los rodea.

1.2.1 Arquitectura del Sistema de Percepción

La arquitectura que presenta el SP con que se está trabajando en el CEDIN y que se muestra en la Figura 1.6, proporciona una serie de elementos a tener en cuenta para la realización de la misma [4] y [11]:

- Mediante el objeto *cPerceptionSystem*, perteneciente a la clase controladora, se realiza la actualización de todas las entidades *cDataGatherer.*
- Mediante la clase *cDataGatherer* se puede almacenar la información de percepción sobre el mundo que puede ser detectada por los agentes.

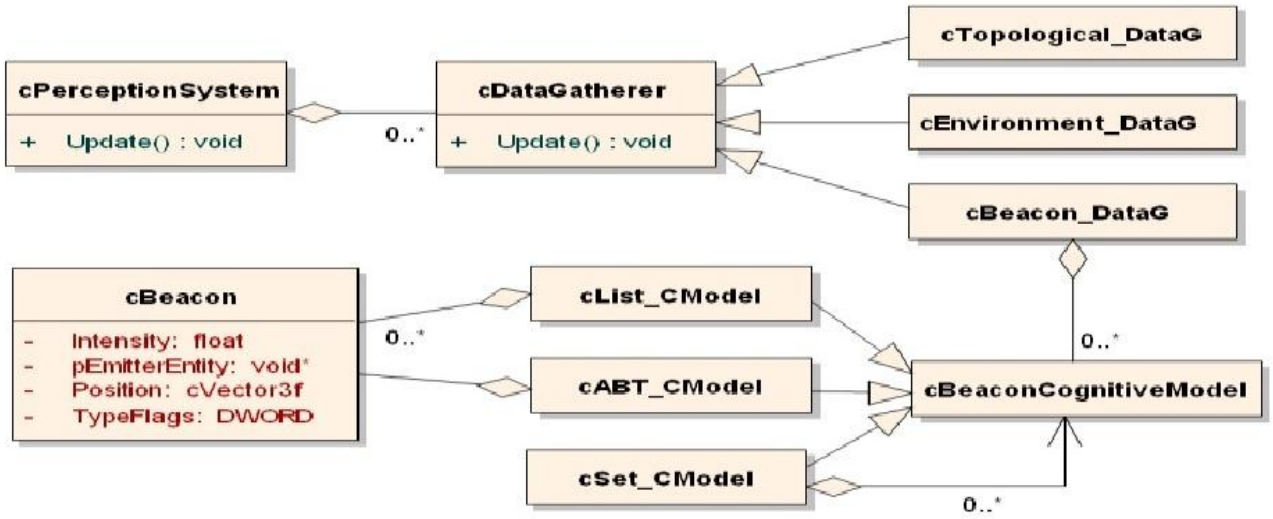


Figura 1.6 Arquitectura del Sistema de Percepción [4] y [11]

- La información detectada puede ser de cualquier tipo como: datos topológicos, datos ambientales y hasta los acontecimientos que suceden en el mundo (*cTopological_DataGatherer, cEnvironment_DataGatherer, cBeacon_DataGatherer,* respectivamente), los cuales serán las clases entidades existentes en el sistema.
- El objeto *cTopological_DataGatherer* es el encargado de buscar la información acerca de las características tácticas del entorno que rodea al agente.

- El objeto *cEnvironment_DataGatherer* es el encargado de recolectar una apreciación del entorno que rodea al agente mediante una representación simplificada del mismo basado en formas básicas.

- Por medio del recolector *cBeacon_DataGatherer* los agentes tienen conocimientos de los objetos dinámicos o eventos que son creados y modificados en tiempo de ejecución. Este recolector, a su vez, está compuesto por uno o más modelos cognitivos que contienen los *beacons* como Lista, ABT, BSP y las rejillas, como modelos más especializados.

- En la clase *cBeacon* se pueden almacenar características relacionadas con el evento dinámico y que son de interés para el agente, como pueden ser el identificador del evento dinámico, la entidad emisora y la intensidad de la emisión.

1.2.2 Componentes de un agente

Por su parte, y para dicha arquitectura con que se trabaja (Figura 1.6), el agente recibe la información del entorno por medio de sensores (Figura 1.7). Los cuales escanean constantemente el entorno en busca de información útil y mantienen al agente actualizado. Para ello, los mismos se conectan a los *DataGatherer* del SP, los cuales le proveen de la información táctica, *beacons* y de estructura del entorno. [4] y [11]

Cuando un agente es creado, se le registran los sensores que él va a necesitar (Figura 1.7), ya que todos los agentes no necesitan conocer todo tipo de información, por ejemplo, un perro no necesita saber de las informaciones tácticas del entorno.

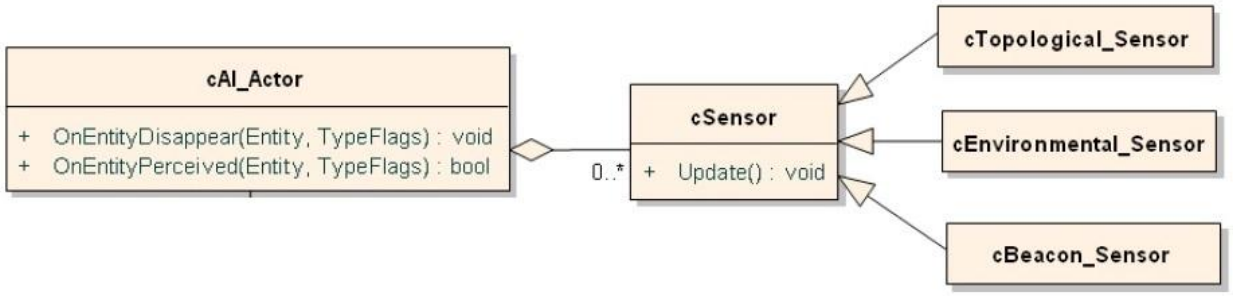


Figura 1.7 Diagrama de clases de los componentes de un agente [4] y [11]

En la figura anterior se pueden apreciar los componentes por los cuales está formado un agente. Una vez que los sensores activan una búsqueda en los recolectores de datos, éstos le reportan al agente la información acerca del entorno, la cual les permitirá tomar las decisiones necesarias para resolver un problema determinado. [4]

1.3 Consideraciones del capítulo

Para definir una buena propuesta de solución es necesario tener conocimiento de las características de las técnicas que permiten la cooperación entre agentes; así como del funcionamiento de la arquitectura que presenta el SP; pues la arquitectura descrita además de manejar las necesidades básicas de percepción de un agente de inteligencia artificial, también puede ser ampliada para incluir algunas optimizaciones adicionales y capacidades interesantes; como puede ser el intercambio de información entre varios agentes para poder lograr que los objetivos de los mismos se cumplan.

CAPÍTULO 2

PROPUESTA Y DESCRIPCIÓN DE LA SOLUCIÓN

Introducción

En el presente capítulo se ofrece una descripción de las características que presentará el sistema como solución al problema científico planteado. Para ello se hace necesario que sean definidas: la metodología de desarrollo de software, la herramienta de modelado y los lenguajes (de modelado y programación) a utilizar para poder llegar a los resultados esperados, mediante una pequeña descripción de los mismos.

Se abordará, además, acerca de las características que presentará la solución propuesta mediante los flujos de trabajo ingenieriles: modelo del negocio y requerimientos, con sus respectivos artefactos.

2.1 Herramientas y tecnologías a utilizar

Para sentar las bases de la propuesta de solución realizada, se hizo necesario utilizar un conjunto de herramientas y tecnologías las cuales soportarán el desarrollo de la misma.

2.1.1 Metodología de desarrollo de software

La metodología de desarrollo de un software es la guía para realizar un software con calidad; definiendo, en un proyecto de software, quién debe hacer qué, cuándo y cómo debe hacerlo.

Se desarrolla con el objetivo de dar solución a los problemas que existen en la producción del software y cuenta con un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software. [9]

Para regir el proceso de desarrollo de esta investigación, se seleccionó a RUP, llamada así por sus siglas en inglés *Rational Unified Process* (Proceso Unificado de Rational); porque es aplicable a proyectos a largo plazo, intentando reducir la complejidad del software y la realización del mismo.

Esta metodología es capaz de adaptarse a las características y complejidad de cualquier proyecto de software realizando una buena captura de los requisitos y siendo apropiada para aplicar a proyectos complejos.

2.1.2 Herramienta de modelado

Se escoge a Visual Paradigm como herramienta de modelado porque es multiplataforma y potente. Soporta el ciclo de vida completo del desarrollo de software, ayuda a una rápida construcción de aplicaciones de calidad y a obtener un menor coste en las mismas. Permite dibujar todos los tipos de diagramas de clases y generar diagramas desde código.

El trabajo con esta herramienta trae como ventajas:

- Presenta capacidades de ingeniería directa e inversa.
- Es un producto de calidad y altamente usado en la población actual.
- Es un producto fácil de instalar, actualizar y manipular.

2.1.3 Lenguaje de modelado

Se selecciona el lenguaje de modelado UML (Lenguaje Unificado de Modelado) porque es utilizado para visualizar, especificar, construir y documentar los artefactos de un sistema obtenidos a partir de una metodología determinada. Se pueden representar todas las fases de un proyecto informático.

Hacer uso del mismo trae como ventajas:

- La trazabilidad y documentación del proyecto se realiza de una forma ordenada y guiada por los casos de uso.
- Es utilizado por Visual Paradigm como lenguaje de modelado.
- Es el más utilizado a nivel mundial para realizar el modelamiento de los artefactos creados durante el proceso de desarrollo de software para los sistemas orientados a objetos (OO).

2.1.4 Lenguaje de programación

El lenguaje de programación C++ estándar, además de ser un lenguaje multiplataforma, es el lenguaje mayormente utilizado para la realización de las aplicaciones de Realidad Virtual (RV),

pues en este tipo de aplicaciones se manejan grandes volúmenes de datos y el mismo permite un uso óptimo de la memoria y del CPU (Unidad Central de Procesamiento) de la computadora donde se instalará la aplicación desarrollada, es por eso que se decide utilizar el mismo para llevar a cabo la implementación de las clases del diseño.

2.2 Descripción del proceso actual

Actualmente en el SP los agentes toman los datos que necesitan del entorno de forma independiente. Trayendo como consecuencias que en varias ocasiones tengan que realizar el mismo proceso que ya fue realizado con anterioridad por otro agente (redundancia en los cálculos de los algoritmos de percepción), lo cual trae como derivaciones que sea demorado y repetitivo el proceso.

Es por ello que la solución a proponer debe lograr que el proceso se realice lo menos reiterativo posible y que logre, a su vez, disminuir los tiempos de los cálculos de los algoritmos de percepción que existen actualmente.

Para ello se hizo un análisis de las técnicas descritas y analizadas en el capítulo anterior.

2.3 Análisis de las técnicas de comunicación entre agentes

Estas técnicas presentan beneficios e inconvenientes entre sí, y se analizan detalladamente para determinar cuál(es) sería(n) la(s) más eficiente(s) y que sea(n), a su vez, la base de construcción de dicho sistema de intercambio de información entre los agentes.

- Los Sistemas de Pizarra son útiles cuando el problema a resolver es muy grande y se necesita que sea compartido entre los demás agentes. Garantiza que una vez encontrada la solución, ésta sea la más factible de todas. [12]

- Los Sistemas de mensaje/diálogo son muy útiles cuando el problema a resolver es muy pequeño y no es necesaria la interacción entre varios agentes. Pero tiene como inconveniente que los demás agentes no pueden tener acceso a la información, a menos que ésta sea dirigida específicamente a ellos.

- Los Sistemas Basados en Disparadores son muy eficientes y rápidos. Presentan cierta complejidad a la hora de ser implementados y con ellos se garantiza encontrar siempre una solución. [12]

- El uso de las MEF se hace efectivo, pues son sistemas muy fáciles de implementar. Presentan como inconveniente que antes de ser aplicados hay que verificar si el problema puede ser descompuesto, es decir, que todos los estados, transiciones y condiciones deben ser conocidos y estar bien definidos con anterioridad.

A partir de las ventajas y desventajas analizadas de dichos sistemas se obtuvo una tabla de comparación (Tabla 2.1) teniendo en cuenta los siguientes parámetros:

- Eficiencia respecto al tiempo de cómputo necesario para resolver el problema.
- Rapidez en la ejecución.
- Garantía de una solución final factible.

Esta comparación se realiza con el objetivo de aportar y justificar los elementos necesarios para seleccionar la(s) técnica(s) por la(s) que se regirá esta investigación.

Técnicas	Eficiencia	Rapidez en la ejecución	Garantía de una solución factible
Sistemas de Pizarra	Si	Si	Si
Sistemas de mensaje/diálogo	Si	Si	No
Sistemas Basados en Disparadores	Si	Si	Si
MEF	Si	Si	Si

Tabla 2.1 Comparación entre las técnicas que permiten el intercambio de información

- En el análisis de la **eficiencia** se puede observar que todas las técnicas cumplen con este parámetro, lo que implica una mejor optimización en el resultado final.
- En cuanto a la **rapidez en la ejecución** se puede observar que todas las técnicas cumplen con este aspecto.
- Los Sistemas de Pizarra, los Sistemas Basados en Disparadores y las MEF **garantizan una solución final factible**, mientras que los Sistemas de mensaje/diálogo se desconoce la solución final porque no pueden intercambiar con varios agentes a la misma vez imposibilitando encontrar la solución en varias ocasiones.

El análisis de los parámetros presentados en la Tabla 2.1, evidencian que las técnicas candidatas para la propuesta de solución de la presente investigación son: Sistemas de Pizarra, Sistemas Basados en Disparadores y las Máquinas de Estados Finitos. Lo cual no significa que la técnica restante no brinde facilidades de uso para los desarrolladores; sólo que ésta no responde debidamente a los criterios tomados en cuenta para la solución a proponer.

2.4 Descripción de los posibles resultados

Se necesita lograr que los agentes no tengan que acceder todos a la información que el entorno les brinda, sino que un agente en específico sea el encargado de tomar la misma y compartirla con sus demás compañeros. Permitiendo, de esta manera, que exista cooperación entre todos y que se logren reducir los tiempos de los algoritmos de percepción que existen actualmente.

Logrando a su vez, que el proceso se realice de forma rápida y que no sea repetitivo.

2.5 Descripción de la solución propuesta

Analizadas las características de las técnicas idóneas y la arquitectura que presenta el SP, se llegan a las siguientes conclusiones:

- En la arquitectura que muestra la Figura 2.1, se puede observar que la información es almacenada en los diferentes modelos cognitivos[1] que existen, es decir, en un único lugar; permitiendo que todos los agentes tengan que acceder al mismo sitio para poder obtener los datos que necesiten del entorno. Se llega a observar entonces, que la arquitectura del SP, brinda soporte para que la técnica Sistemas de Pizarra sea implementada.

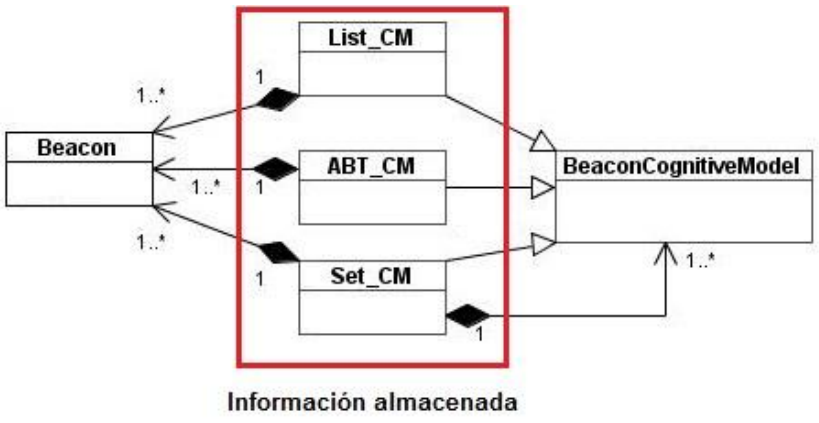


Figura 2.1 Información almacenada

[1] Son los encargados de brindarle a los agentes los datos del entorno.

- Anteriormente, en el Capítulo 1 en el epígrafe 1.1.4.3 Sistemas Basados en Disparadores (*Trigger*), se describió que un disparador o *trigger* puede ser cualquier estímulo o evento que el diseñador de la aplicación decida que el agente pueda responder a él.

  Analizada esta descripción, se puede decir que un disparador es el equivalente, en la arquitectura del SP antes expuesta; a los *beacons* que en ella se describen, lo que permite admitir que dicha arquitectura presenta también, características, para que la técnica Sistemas Basados en Disparadores o *Trigger* sea implementada.

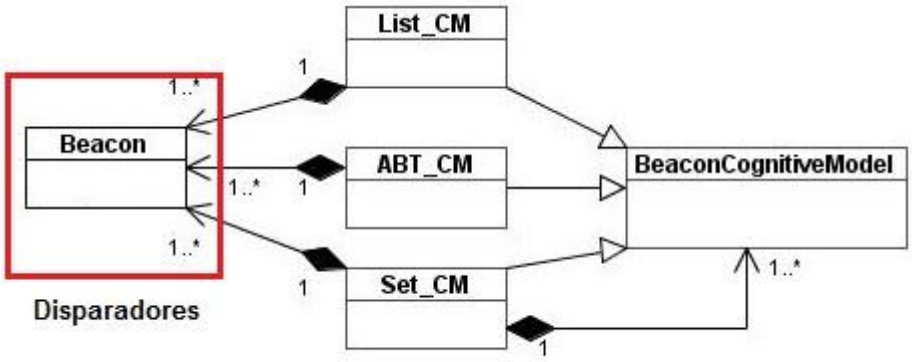


Figura 2.2 *Beacon*, equivalente a Disparadores

Se decide entonces, realizar un híbrido entre las técnicas Sistemas de Pizarra y Sistemas Basados en Disparadores, que posibilite crear grupos de agentes que estarán encabezados por un líder y éste les brindará los datos que necesiten del entorno. La cual contendrá dentro de sus funcionalidades principales:

- Adicionar y eliminar agentes de un grupo.
- Registrar los datos de interés de cada grupo.
- Crear, actualizar y eliminar los grupos de agentes.

La arquitectura propuesta en la Figura 2.3, extenderá las funcionalidades de la arquitectura inicial a partir de la adición de las siguientes clases:

- ***PSGroupAgentController:*** Se encargará de tener un listado con todos los grupos de agentes que existen en el entorno. Tendrá un control total sobre estos grupos, ya que es la encomendada de adicionar los grupos que sean creados, eliminarlos si es necesario y actualizarlos cuando sea necesario.

- ***PSGroupAgent:*** Se encargará de administrar los agentes por los cuales estará conformado un grupo a partir de los datos de interés que a éste le hayan sido definidos. Heredará de la clase *AgentPerceptionSystem* porque los agentes que integran los grupos serán de dicho tipo, y a su vez, tendrá una relación denominada patrón *composite*[2] con dicha clase, permitiendo almacenar un listado recursivo de todos los agentes del entorno, para hacer más fácil el trabajo con los mismos.

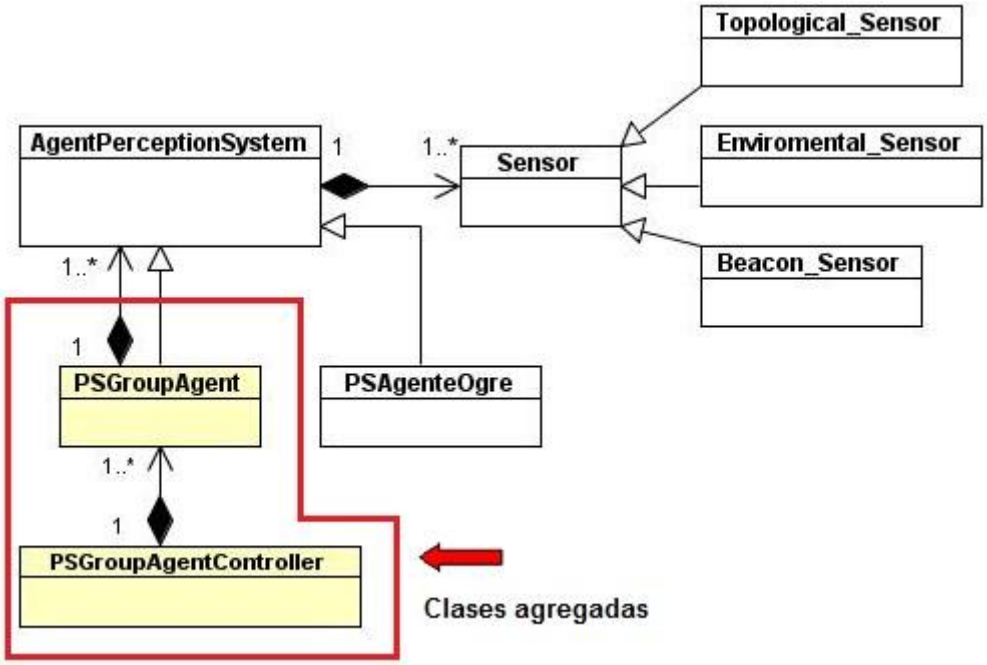


Figura 2.3 Clases agregadas (sin métodos)

2.6 Modelamiento del negocio

El flujo de trabajo modelamiento del negocio es uno de los flujos que mayor peso tiene en la fase de inicio.

Este flujo presenta dentro de sus objetivos: derivar los requerimientos del sistema, quienes serán el soporte para lograr la construcción de la solución propuesta.

Como no se hace necesario realizar el modelo completo del negocio, ya que la documentación que existe no es suficiente para conocer los diferentes procesos por los que pasa el sistema a desarrollar, se realizará lo que se conoce como modelo del dominio.

[2] Consultar el glosario de términos para un mejor entendimiento.

2.6.1 Modelo del dominio

El modelo del dominio captura los tipos de objetos más importantes que existen o los eventos que suceden en el entorno donde estará el sistema. Por ello, a continuación se representa un acercamiento a la solución propuesta, donde se modelan los principales conceptos con los que se trabajarán en el desarrollo de la solución, así como las relaciones existentes entre ellos.

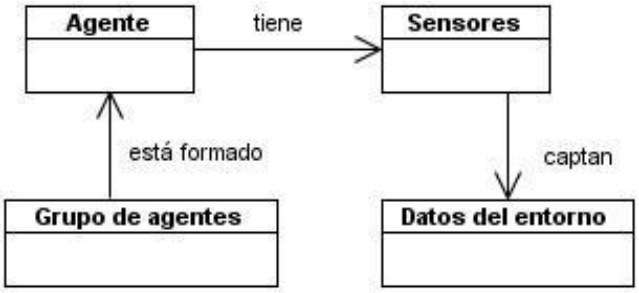


Figura 2.4 Modelo del dominio

2.6.2 Glosario de términos

A continuación se relacionan los conceptos presentes en el modelo del dominio anterior con sus respectivas especificaciones para facilitar la comprensión del mismo:

Agente: Es la entidad inteligente por la cual estará conformado el SP, es capaz de percibir el entorno que lo rodea, interactuar con él y con otros agentes; y tomar las decisiones que sean necesarias para darle solución a los problemas u objetivos que se le puedan presentar.

Sensores: Serán los encargados de otorgarles a los agentes la información básica del entorno que éstos necesitan para resolver sus problemas.

Grupo de agentes: Estará compuesto por varios agentes del sistema con intereses comunes de percepción.

Datos del entorno: Son los *DataGatherer* del SP, pues en ellos se puede almacenar la información de percepción sobre el mundo que puede ser detectada por los agentes.

2.7 Requerimientos del sistema

Los requerimientos constituyen la capacidad o término que necesita un usuario para resolver un problema y que debe ser alcanzada por un sistema para satisfacer un contrato u otro documento impuesto formalmente. [7]

Éstos son clasificados en requisitos funcionales y requisitos no funcionales, indicando las funcionalidades e instrucciones que el sistema debe cumplir en su implementación y las propiedades o cualidades que el sistema debe tener, estableciendo, de esta forma; aspectos que regulan el comportamiento del mismo, respectivamente.

2.7.1 Requisitos funcionales

A continuación se muestra una descripción detallada, para lograr un mejor entendimiento, de los requisitos funcionales (RF).

Nº	Nombre	Descripción	Complejidad	Prioridad para el cliente
RF1	Crear los grupos de agentes.	El sistema debe ser capaz de crear los grupos de agentes de forma manual, de acuerdo a los datos de interés que tengan los mismos especificados.	Alta	Alta
RF2	Actualizar los grupos de agentes.	El sistema debe ser capaz de actualizar los grupos de agentes, de forma que los datos que éstos reciban del entorno sean proporcionados por un sólo agente.	Alta	Alta
RF3	Eliminar los grupos de agentes.	El sistema debe ser capaz de eliminar los grupos de agentes, de acuerdo a un grupo pasado por parámetro.	Media	Alta
RF4	Registrar los datos de interés de los grupos.	El sistema debe ser capaz de registrarle a cada uno de los grupos de agentes creados, los datos de interés que tendrán.	Baja	Alta

RF5	Adicionar agentes a un grupo.	El sistema debe ser capaz de adicionar agentes a un grupo específico, permitiendo integrar los mismos al trabajo con el entorno.	Media	Alta
RF6	Eliminar agentes de un grupo.	El sistema debe ser capaz de eliminar agentes de un grupo específico, permitiendo que éstos sean desvinculados.	Media	Alta

Tabla 2.2 Especificación de los requisitos funcionales

2.7.2 Requisitos no funcionales

Los requisitos no funcionales (RnF) fueron detallados según el plan de mejora que se usa en la UCI. A continuación se muestran las especificaciones de cada uno de ellos:

a) RnF1. <Requisito de Usabilidad>

- **Finalidad:** La aplicación tiene como objetivo gestionar, de manera continua, el proceso de cooperación entre los agentes de un determinado entorno virtual. Evitando a su vez, que existan redundancias en los cálculos de los algoritmos de percepción que existen actualmente.

b) RnF2. <Requisito de Soporte>

- **Soporte:** El sistema que se desarrollará, en su finalidad, deberá tener compatibilidad con el sistema operativo Windows y cualquier distribución de Linux.
- **Hardware:** Se necesita que la computadora, en que se vaya a utilizar la aplicación, conste de un Microprocesador Intel Pentium 4 a 1.00GHz. Y que tenga una memoria RAM de 512MB DDR2, como mínimo.

c) RnF3. <Requisito de Restricciones del Diseño>

- **Lenguaje de programación:** Se utilizará el lenguaje de programación C++ estándar bajo el paradigma de programación OO.

d) RnF4. <Requisito de Estándares Aplicables>

- **Estándares de codificación:** Se utilizarán los estándares de codificación definidos por la Línea de navegación y comportamiento inteligente del CEDIN (Ver Anexo 2).

2.8 Modelo de caso de uso del sistema

En esta sección se reconocen los actores del sistema a desarrollar, y se definen los CU (caso de uso) del sistema. Además, se seleccionan los CU correspondientes al primer ciclo de desarrollo creando sus especificaciones textuales en formato expandido.

2.8.1 Actores del sistema

Los actores de un sistema son agentes externos o roles que las personas o usuarios desempeñan cuando interactúan con el software que se realizará. [6]

En el caso particular de esta investigación, quien inicializará el sistema será la misma aplicación en sí, que como actor del sistema será llamado Aplicación.

Actor	Descripción
Aplicación	Es el encargado de inicializar el sistema permitiendo que sean ejecutadas las funcionalidades: crear, actualizar y eliminar grupos de agentes; registrar a los grupos los datos de interés, adicionar los agentes y eliminar los mismos de los grupos de agentes.

Tabla 2.3 Actor del sistema

2.8.2 Diagrama de CU del sistema

El diagrama de CU del sistema define las relaciones entre los actores y los diferentes CU existentes.

Para la realización de esta investigación fueron definidas un conjunto de acciones que deben ser ejecutadas por el actor del sistema y que desencadenan un conjunto de operaciones. Las interrelaciones entre las acciones y el actor del sistema son agrupadas en el diagrama de CU del sistema que se muestra a continuación.

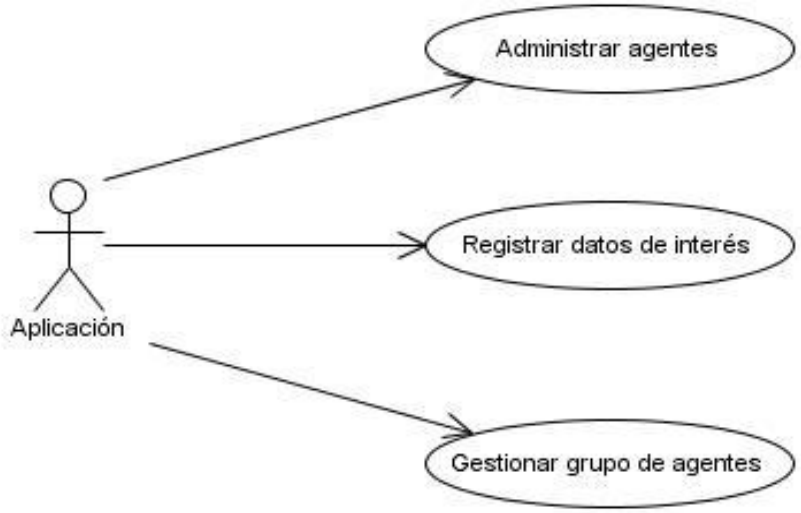


Figura 2.5 Diagrama de CU del sistema

2.8.3 Descripción de los CU del sistema

Los CU son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario, estableciendo un acuerdo entre clientes y desarrolladores sobre las condiciones y posibilidades (requisitos) que debe cumplir el sistema. [6]

Cada CU tiene una descripción de las funcionalidades que ejecutará el sistema propuesto como respuesta a las acciones del usuario. Las descripciones presentadas a continuación muestran detalladamente los flujos operacionales de éstos.

CU 1	Gestionar grupo de agentes
Actores	Aplicación
Resumen	El CU inicia cuando la aplicación necesita tener un control sobre los grupos de agentes que existen en el entorno, permitiendo que éstos sean creados, actualizados y eliminados.
Precondiciones	Debe, al menos, existir un agente en el entorno.
Referencias	RF1, RF2, RF3

Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. La Aplicación llama a las diferentes funcionalidades: a) Crear grupos de agentes. b) Actualizar grupos de agentes. c) Eliminar grupos de agentes.	1.1 El sistema inicializa las secciones “Crear grupos de agentes” “Actualizar grupos de agentes” y “Eliminar grupos de agentes”.
Sección “Crear grupos de agentes”	
Acción del Actor	**Respuesta del Negocio**
2. La Aplicación pasa los siguientes datos: a) Datos de interés por los que estarán formados los grupos. b) El radio de vecindad que tengan en el entorno que los rodea.	2.1 El sistema crea los grupos de agentes de acuerdo a los datos recibidos y finaliza el CU.
Sección “Actualizar grupos de agentes”	
Acción del Actor	**Respuesta del Negocio**
3. La Aplicación manda a actualizar al líder de los grupos de agentes.	3.1 El líder toma los datos que el entorno le ofrece. 3.2 Para los datos tomados a sus demás compañeros y finaliza el CU.
Sección “Eliminar grupos de agentes”	
Acción del Actor	**Respuesta del Negocio**

4. La Aplicación pasa el número del grupo que se desea eliminar.	4.1 El sistema busca el grupo que se desea eliminar. 4.2 El grupo es eliminado y finaliza el CU.
Flujo Alterno de Eventos	
Sección "Eliminar grupos de agentes"	
Acción del Actor	**Respuesta del Negocio**
	4.1 El grupo no existe. 4.2 Muestra un error y finaliza el CU.
Poscondiciones	Quedan creados, actualizados o eliminados los grupos de agentes del entorno.

Tabla 2.1 CU Gestionar grupo de agentes

CU 2	Registrar datos de interés
Actores	Aplicación
Resumen	El CU inicia cuando la aplicación pasa por parámetro un grupo de agentes. Al mismo se le registran los datos que son de interés para sí.
Precondiciones	Debe, al menos, existir un grupo de agentes en el entorno.
Referencias	RF4
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. La Aplicación pasa por parámetro el grupo de	1.1 El sistema busca el grupo de agentes

agentes y los datos de interés del mismo.	recibido. 1.2 Registra los datos recibidos por parámetro y finaliza el CU.
Flujo Alterno de Eventos	
Acción del Actor	**Respuesta del Negocio**
	1.1 El grupo de agentes no existe. 1.2 Muestra un error y finaliza el CU.
Poscondiciones	Quedan registrados los datos de interés del grupo de agentes seleccionado.

Tabla 2.2 CU Registrar datos de interés

CU 3	Administrar agentes
Actores	Aplicación
Resumen	El CU inicia cuando la aplicación necesita adicionar y eliminar determinado agente a un grupo.
Precondiciones	Debe, al menos, existir un grupo de agentes en el entorno.
Referencias	RF5, RF6
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Negocio**
1. La Aplicación llama a las diferentes funcionalidades: d) Adicionar agente.	1.1 El sistema inicializa las secciones “Adicionar agente” y “Eliminar agente”.

e) Eliminar agente.	
Sección "Adicionar agente"	
Flujo Normal de Eventos	**Flujo Normal de Eventos**
2. La Aplicación pasa por parámetro el agente a adicionar.	2.1 El sistema busca en su listado de grupos, el que tenga los mismos datos de interés que el agente recibido. 2.2 Cuando lo encuentra, adiciona el agente a dicho grupo, finalizando así el CU.
Sección "Eliminar agente"	
Acción del Actor	**Respuesta del Negocio**
3. La Aplicación para por parámetro el número del agente que necesita eliminar.	3.1 El sistema busca en su listado de grupos el agente a eliminar. 3.2 Una vez encontrado, lo elimina del grupo en que se encuentre, finalizando así el CU.
Flujo Alterno de Eventos	
Sección "Adicionar agente"	
Acción del Actor	**Respuesta del Negocio**
	2.1 No existe el grupo con los mismos datos de interés. 2.2 Muestra un error y finaliza el CU.
Flujo Alterno de Eventos	
Sección "Eliminar agente"	
Acción del Actor	**Respuesta del Negocio**

	3.1 No existe en el grupo el agente a eliminar. 3.2 Muestra un error y finaliza el CU.
Poscondiciones	El agente queda adicionado o eliminado de un grupo.

Tabla 2.3 CU Administrar agentes

2.9 Consideraciones del capítulo

En el capítulo que aquí concluye, se detalló la solución propuesta, quedando sentadas las bases sobre las cuales se podrá iniciar la implementación del sistema haciendo uso de las herramientas seleccionadas.

Se decide, además, realizar un híbrido entre las técnicas Sistemas de Pizarra y Sistemas Basados en Disparadores, que permita la creación de grupos en el entorno los cuales estarán guiados por un líder. Siendo éste el encargado de transmitirle a sus compañeros los datos del entorno.

CAPÍTULO 3

DISEÑO, IMPLEMENTACIÓN Y RESULTADOS

Introducción

En este capítulo se abordan los temas del diseño y de la implementación del sistema, basado en todo el trabajo acumulado a lo largo del desarrollo de los capítulos anteriores y mostrando la realización de los distintos artefactos correspondientes a cada uno de los flujos de trabajo analizados; así como el análisis de los resultados obtenidos al desarrollar la aplicación que dará cumplimiento a los objetivos planteados con anterioridad.

3.1 Modelo del diseño

El modelo del diseño describe la realización física de los CU centrándose tanto en los requisitos funcionales como en los requisitos no funcionales.

En el diseño se modela el sistema y se confecciona su estructura (arquitectura), que sirve de soporte para la realización de todos los requisitos.

La realización de los CU del diseño contiene una descripción de los flujos de eventos textuales, diagramas de clases y diagramas de interacción. Los diagramas de clases son los más utilizados en el modelado de sistemas OO. [6]

3.1.1 Diagrama de clases del diseño

Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como las relaciones existentes entre las mismas. A una clase del diseño, en la implementación del sistema, se le atribuyen visibilidad a sus atributos y operaciones.

A continuación se encuentra el diagrama de clases del diseño del SP (Figura 3.1) correspondiente a los sensores, pero sólo con los métodos y atributos en las clases que fueron añadidas en la propuesta de solución. Para lograr una mejor comprensión de las relaciones existentes entre dichas clases, es necesario que sean consultados los Anexos 3 y 4, propios de los diagramas de

clases del SP y al de las clases agregadas respectivamente, con sus métodos y atributos completos.

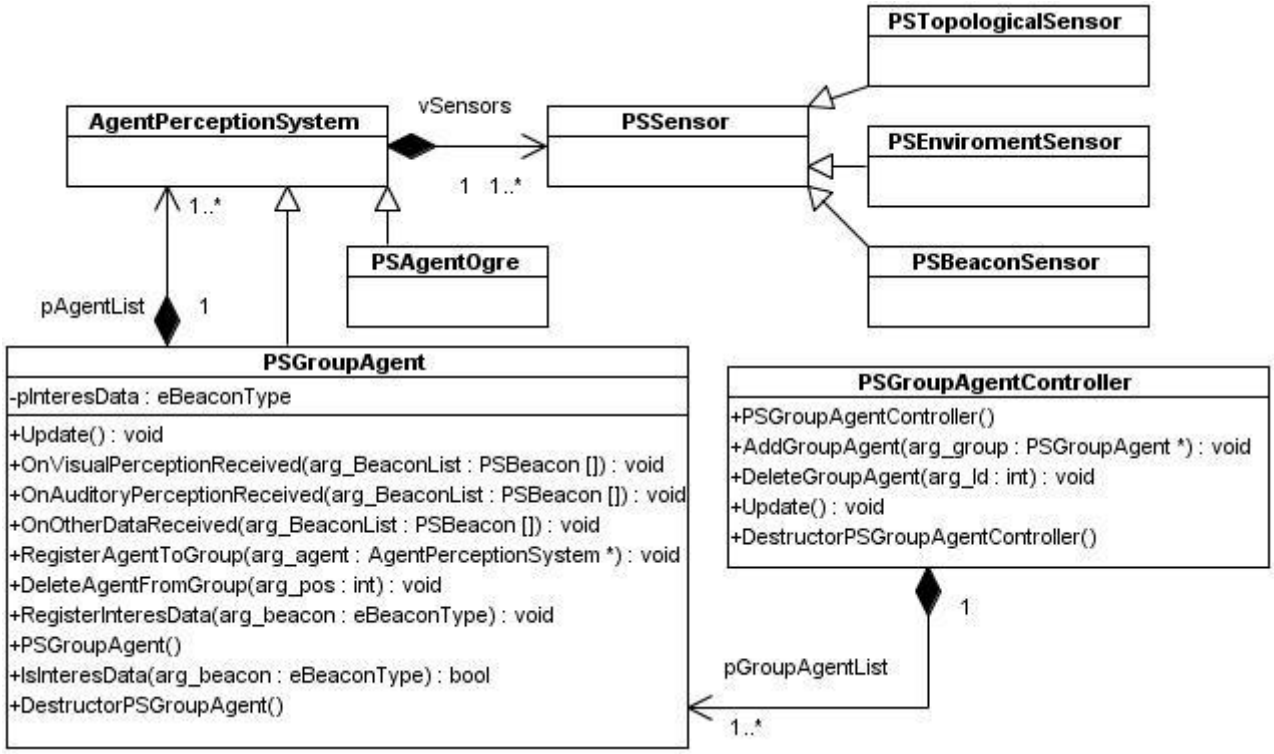


Figura 3.1 Diagrama de clases del diseño

3.1.2 Diagramas de secuencia del diseño

A continuación se encuentran los diagramas de secuencia del diseño para cada uno de los distintos CU y clases existentes en dicho sistema; de forma tal que se facilite la comprensión de las relaciones entre los mismos.

3.1.2.1 CU Gestionar grupo de agentes. Sección Crear grupos de agentes

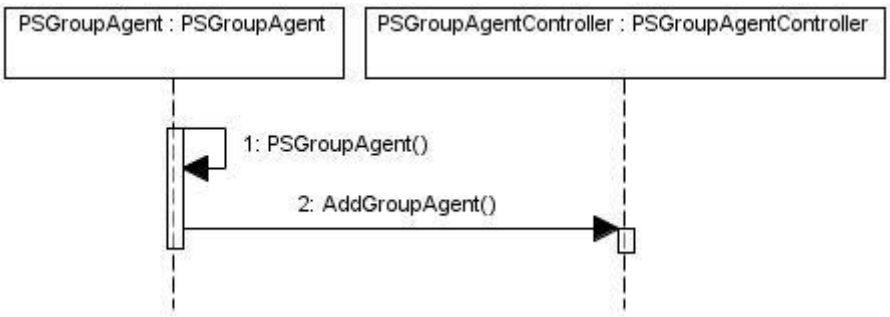


Figura 3.2 Diagrama de secuencia CU Gestionar grupo de agentes. Sección Crear grupos de agentes

3.1.2.2 CU Gestionar grupo de agentes. Sección Actualizar grupos de agentes

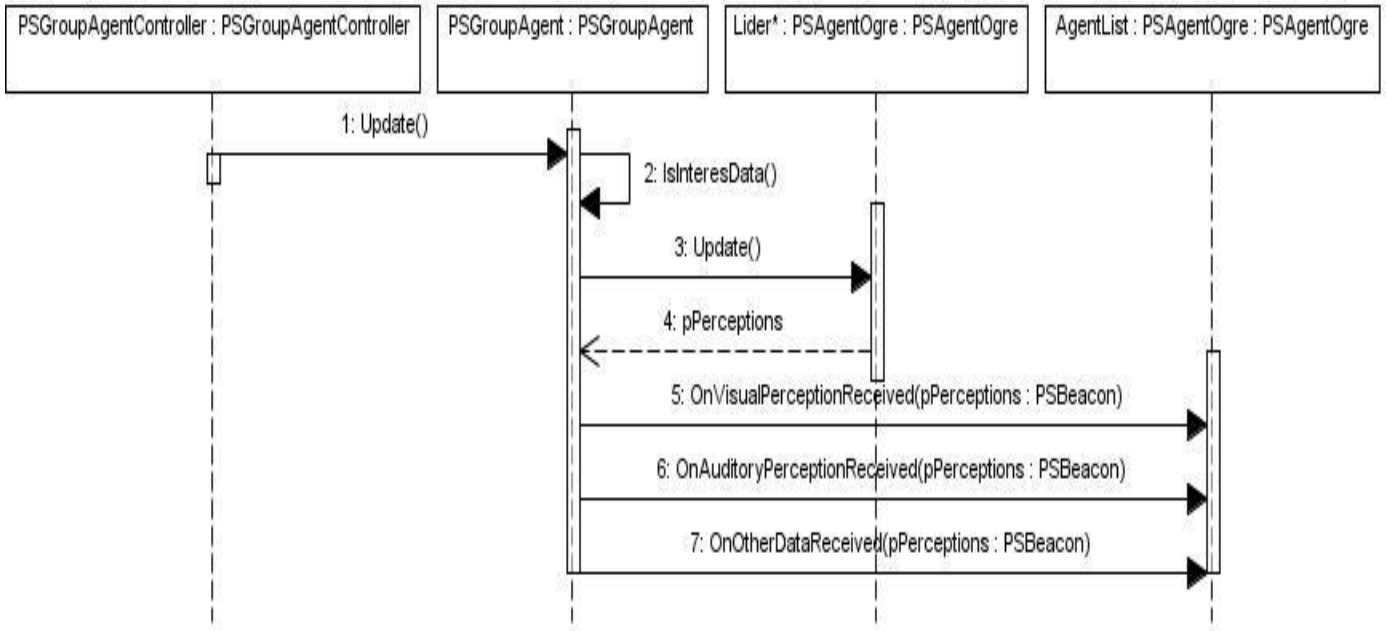


Figura 3.3 Diagrama de secuencia Gestionar grupo de agentes. Sección Actualizar grupos de agentes

3.1.2.3 CU Gestionar grupo de agentes. Sección Eliminar grupos de agentes

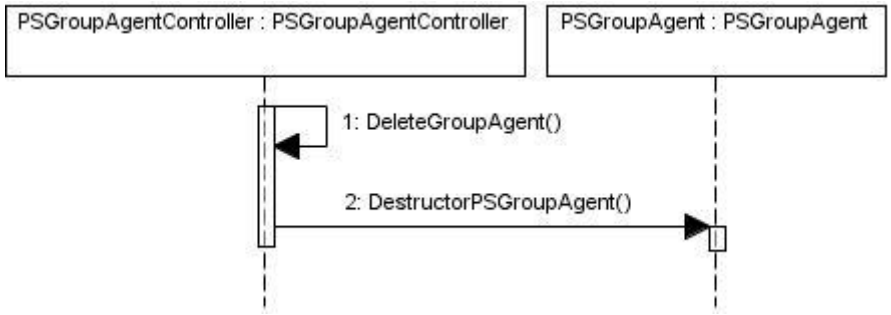


Figura 3.4 Diagrama de secuencia CU Gestionar grupo de agentes. Sección Eliminar grupos de agentes

3.1.2.4 CU Registrar datos de interés

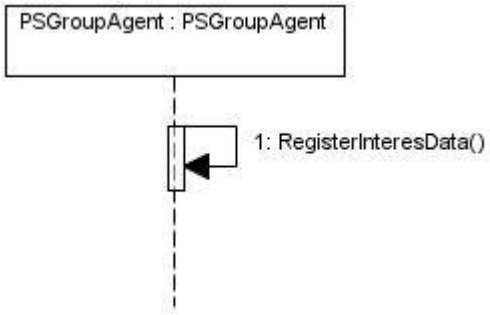


Figura 3.5 Diagrama de secuencia CU Registrar datos de interés

3.1.2.5 CU Administrar agentes. Sección Adicionar agente a un grupo

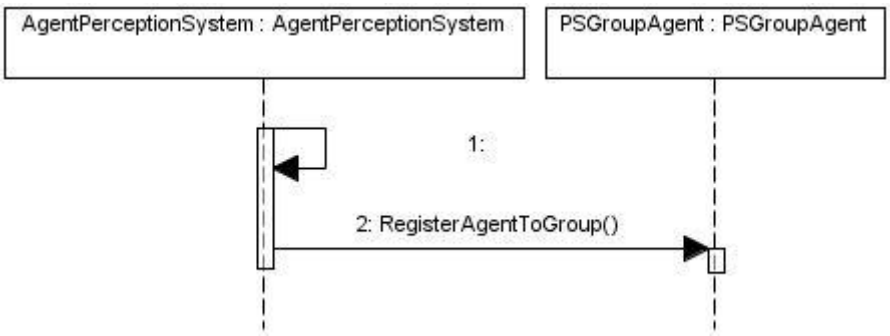


Figura 3.6 Diagrama de secuencia CU Adicionar agente a un grupo

3.1.2.6 CU Administrar agentes. Sección Eliminar agente de un grupo

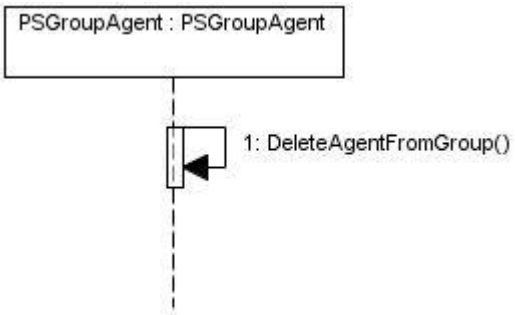


Figura 3.7 Diagrama de secuencia CU Eliminar agente de un grupo

3.2 Modelo de implementación

El flujo de trabajo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo éstos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

Los diagramas de componentes y de despliegue, que son artefactos generados en este flujo de trabajo, conforman lo que se conoce como un modelo de implementación, al describir los componentes a construir y su organización; así como su dependencia entre nodos físicos en los que funcionará la aplicación, respectivamente. [6]

3.2.1 Diagrama de componentes

Las clases resultantes del diseño se hacen físicas mediante componentes. A continuación se muestra cómo se agruparon dichas clases en los componentes según las relaciones que tienen

entre sí y una breve descripción de las mismas, entre los componentes que más se utilizan para el desarrollo de esta aplicación.

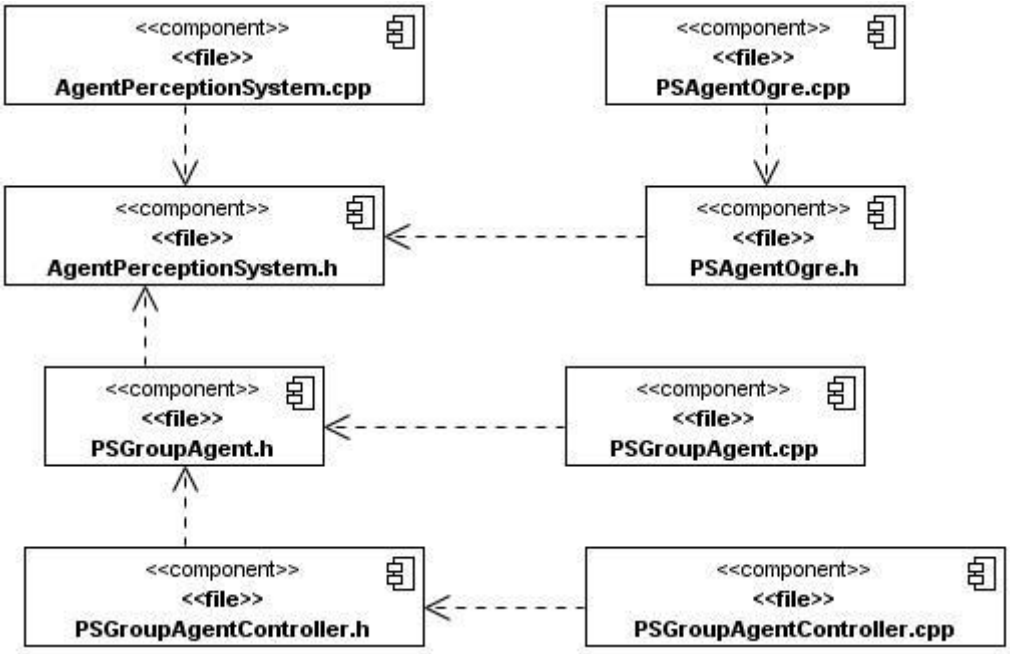


Figura 3.8 Diagrama de componentes

Para que el componente *PSGroupAgentController* pueda realizar su trabajo y cumplir con la realización de las funcionalidades que le corresponde, necesita tener incluido al componente *PSGroupAgent* y este último, necesita del componente *AgentPerceptionSystem*, para poder cumplir con sus objetivos y tareas.

3.2.2 Diagrama de despliegue

En el diagrama de despliegue se indica la situación física de los componentes lógicos desarrollados. Es decir, se sitúa el software en el hardware que lo contendrá una vez terminada la realización del mismo.

Cada hardware se representa como un nodo. Un nodo se representa como un cubo y es un elemento donde se ejecutan los componentes desarrollados. [6]

Atendiendo a las características del sistema, antes planteadas, se realizó el diagrama de despliegue que se muestra a continuación, el cual satisface las necesidades y exigencias de la aplicación, demostrando que la aplicación realizada será desplegada en una computadora, lo que permitirá el uso de la misma por parte de los usuarios finales.

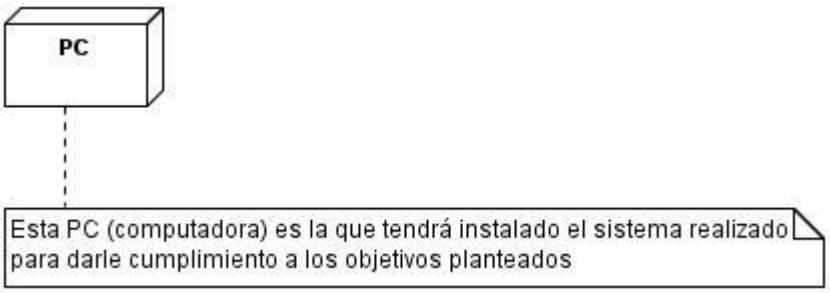


Figura 3.9 Diagrama de despliegue

3.3 Descripción de los resultados obtenidos

Se obtuvo una aplicación que se comenzó a utilizar en los escenarios del proyecto productivo Entrenadores Aduaneros, el cual se desenvuelve en el Aeropuerto Internacional José Martí, ubicado en La Habana, Cuba. Con la simulación de un vuelo que arriba a dicho aeropuerto y que contiene, dentro de sí, un número limitado de pasajeros.

Estos pasajeros son los agentes inteligentes y con ellos se crearon grupos de agentes, los cuales estaban encabezados por un líder. Este líder fue el encargado de interactuar con el entorno y tomar del mismo los datos que necesitaba. Una vez tomados dicho datos, se los transmitía a sus demás compañeros; logrando la cooperación entre todos los agentes del entorno y que las redundancias en los cálculos de los algoritmos de percepción que antes existían, hayan sido erradicadas al realizar el módulo de cooperación.

Para ello fue necesario hacerle pruebas de rendimiento a dicho módulo, las cuales permitirán validar el mismo.

3.4 Resultados

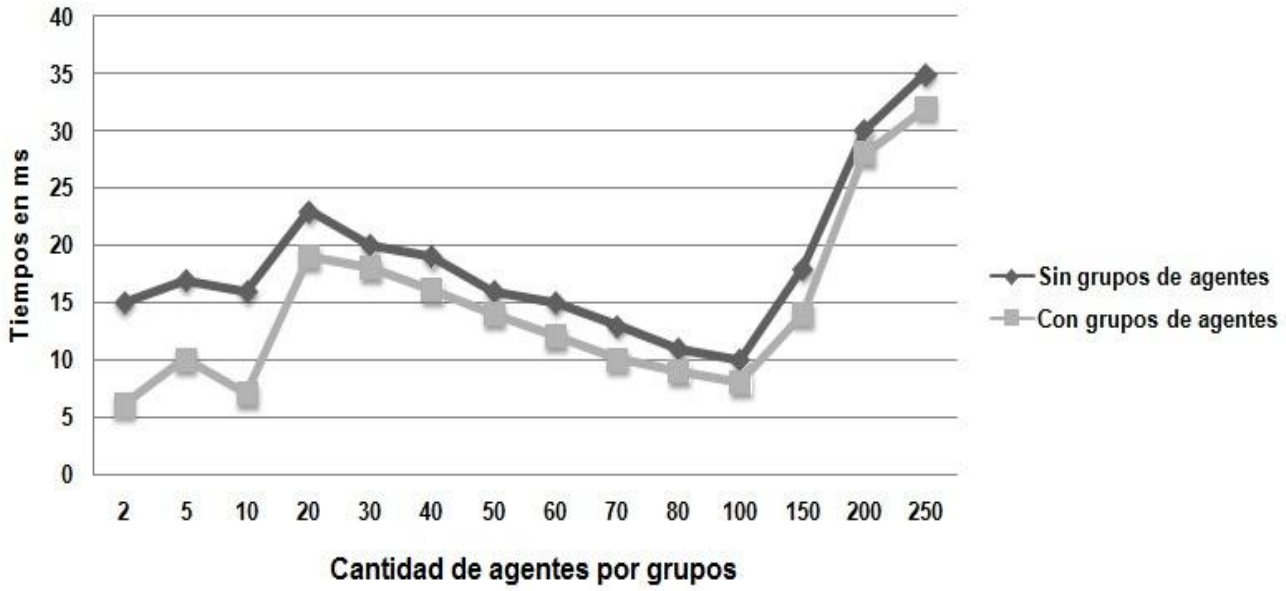
Las pruebas de rendimiento, arrojaron cierta cantidad de valores de tiempo expresados en milisegundos (Ms). Los mismos fueron analizados y registrados en la Tabla 3.1, mostrando los peores tiempos de los cálculos de los algoritmos que existían antes de crear los grupos de agentes y los que existen actualmente con la creación de los mismos.

Para ello fueron analizados un total de 14 grupos de agentes, y a éstos se les asignaron cantidades distintas de números de agentes para poder realizar comparaciones.

Los valores obtenidos se representan en la Gráfica 3.1 y permiten arribar a las observaciones que más adelante se detallan.

No. Agentes	Tiempo sin grupos en Ms (TSG)	Tiempo con grupos en Ms (TCG)	No. Agentes	Tiempo sin grupos en Ms (TSG)	Tiempo con grupos en Ms (TCG)
2	15	6	**60**	15	12
5	17	10	**70**	13	10
10	16	7	**80**	11	9
20	23	19	**100**	10	8
30	20	18	**150**	18	14
40	19	16	**200**	30	28
50	16	14	**250**	35	32

Tabla 3.1 Tiempos en ms (con grupos y sin grupos)



Gráfica 3.1 Representación gráfica de los tiempos obtenidos

Observaciones:

- El tiempo obtenido, al ser creados los grupos de agentes, logra reducir las redundancias en los cálculos de los algoritmos de percepción que antes existían.
- El 50% de los grupos mantiene una posición estable de tiempo y se va reduciendo el mismo a medida en que van aumentando los números de agentes, comportándose de mejor manera con la creación de los grupos de agentes, que sin ellos.
- A partir del 78.57 %, aumentan notablemente los cálculos de los algoritmos de percepción, por lo que no es recomendable utilizar dicha aplicación para entornos que excedan los 200 agentes.

3.5 Consideraciones del capítulo

En el capítulo que concluye se realizaron los flujos de trabajo correspondientes al diseño y a la implementación, generando cada uno de los artefactos proporcionados por la metodología seleccionada y por las características principales de la propuesta realizada. Se realiza un análisis de los resultados obtenidos y se describe detalladamente la aplicación desarrollada.

CONCLUSIONES

Al concluir la investigación realizada y después de haber valorado cada uno de los resultados obtenidos en los capítulos anteriores, se puede arribar a las siguientes conclusiones:

- Se desarrolló una nueva técnica de cooperación, resultado de unir las principales características de las técnicas Sistemas de Pizarra y Sistemas Basados en Disparadores, que sirvió para sentar las bases y poderle dar cumplimiento al objetivo general formulado.
- La arquitectura propuesta, mejora las características que presenta la inicial, logrando reducir los tiempos en los cálculos de los algoritmos de percepción que antes existían.
- La extensión realizada, con la agregación de las nuevas clases, mejora las funcionalidades de la arquitectura inicial. Quedan desarrolladas las características: crear grupos de agentes, actualizarlos y eliminarlos; registrar agentes a un grupo y eliminarlos, así como registrar los datos de interés a cada grupo; permitiendo que sea acoplada al SP.
- El módulo obtenido se probó en el proyecto Entrenadores Aduaneros y estará disponible para ser usado por cualquier sistema que lo necesite.

RECOMENDACIONES

Se recomienda:

- Desarrollar la creación de los grupos de agentes de forma dinámica.
- Extender la propuesta realizada a la cooperación entre agentes de los Sistemas Multi-Agentes.

BIBLIOGRAFÍA REFERENCIADA

[1] Botti, V. *Agentes inteligentes: El siguiente paso en la Inteligencia Artificial.* 2000.

[2] Caglayan, Ana. *Agent Sourcebook.* 1997.

[3] Defensa, VD. *Proceso de Desarrollo y gestión de Proyectos de Software (1ra versión).* Ciudad Habana : s.n., 2009.

[4] Del Valle Guevara, Yenifer. *Sistema de percepción genérico para agentes autónomos en videojuegos.* Universidad de las Ciencias Informáticas : s.n., 2010.

[6] Jaconson, Ivar y otros. *El Proceso Unificado de Desarrollo de Software.* 2000.

[7] Kruchten, P y Wesley, Addison. *The Rational Unified Process: An Introduction.* 2000.

[8] Llamas Bello, César. *Introducción a los Agentes y Sistemas Multi-Agentes.* Valladolid : s.n., 2000.

[9] Mendoza Sánchez, María A. *Metodología de Desarrollo de Software.* 2004.

[10] Pajuelo Morán, Carlos y Alvarez García, Alvaro. *Inteligencia Artificial? Contra Quién?* s.l. : Ingenierías Industriales, 2007.

[11] Puig Placeres, Frank. Generic Perception System. [aut. libro] Steve Rabin. Ai Game Programming Wisdom 4. s.l. : Charles River Media - Estados Unidos, 2006, págs. 285-294.

[11] Ramón Rizo, Faraón y Pujol Llorens, Mar. *Arquitectura y Comunicación entre agentes.* Universidad de Alicante : s.n., 2008.

[12] Sánchez Bernillo, Haydeé y Valdés Pena, Lisandra. *Propuesta y técnica para la Comunicación entre Agentes Virtuales.* Universidad de las Ciencias Informáticas : s.n., 2008.

[13] Vallejo Díaz, L. *Propuesta de arquitectura para el sistema de gestión automatizado de recursos humanos GESTAPro.* 2009.

BIBLIOGRAFÍA CONSULTADA

[14] Bourg, David M y Seeman, Glenn. *AI for Game develipers.* O'Reilly : s.n., 2004.

[15] Cheol-Min, Kim, y otros. *Designing Architecture for Constructing a Virtual Marine World*. 7A. *International Journal of Computer Science and Network Security, Vol. 6*. Julio de 2006.

[16] Corkill, Daniel D. *AI Expert-Blacboard Systems.* Technology Group.

[17] Curello, P.A. *Simulación de la Dinámica de los Cuerpos Rígidos en Tiempo Real*. Instituto Tecnológico de Buenos Aires.

[18] González, Alfredo y Baquiero, Omar. *Metodologías de la IA (Agentes autónomos y Redes neuronales Supervisadas) aplicadas a NPCs (Non player characters).* 2004

[18] Hernando, Ramón. Logica Difusa. [En línea] http://campusvirtual.unex.es/cala/epistemowikia/index.php?title=L%C3%B3gica_difusa.

[19] Herrero Martín, María del Pilar. *Modelo de percepción para agentes virtuales inteligentes*. Tesis doctoral. Madrid: Universidad politécnica de Madrid. Departamento de informática, 2003.

[20] Jepp, Pauline, Fradinho, Manuel y Madeiras, Johao. *An Agent Framework for a Modular Serious Game*. s.l. : Second International Conference on Games and Virtual Worlds for Serious Applications, 2010.

[21] Larman, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Editorial Félix Varela ed. 2004, La Habana.

[22] Peláez, Orfilio y Martínez, Leticia. *Salto hacia la soberanía tecnológica.* Granma : s.n., 2007.

[23] Pipho, E. y Lamothe, A. *3D Model, Game Development Series*. Cincinnati, Ohio. 2003.

[24] Rabin, Steve. *AI Game Programming Wisdom 1.* 2002.

[25] Rabin, Steve. *AI Game Programming Wisdom 2.* 2002.

[26] Weibin, Liu, y otros. *Modeling Human-like Autonomous Behaviors and Movements of Virtual Humans in Real-time Virtual Environment.* 2010.

GLOSARIO DE TÉRMINOS

Beacon: Es el sinónimo de Disparador o Trigger en la arquitectura del Sistema de Percepción. Puede ser cualquier estímulo, en un juego, que requiera de un agente para responder. En un entorno determinado, los *beacons*, podrían ser cualquier cosa audible o visible, que afecta el comportamiento de los agentes, tales como: disparos de armas de fuego, explosiones, enemigos cercanos, ruidos de las puertas al cerrarse o los cadáveres, etc.

Entorno virtual: Es el encargado de interactuar con el agente, comunicar independencia, colaborar y elegir, dándose la oportunidad de convertirse en un nuevo modelo comunicativo, un poco más interactivo, participativo, divertido y colaborativo. [10]

Híbrido: Cruce de características o cualidades específicas de sus ancestros (padres), permitiendo que sean más idóneos que éstos en su utilización específica.

Modelo cognitivo: Representación del mundo virtual, que ofrece información a los agentes del entorno para que puedan interactuar con el mismo de manera eficiente, haciendo uso de diferentes estructuras de datos: Lista, ABT, BSP y las rejillas, como modelos más especializados.

Patrón composite: Sirve para construir objetos complejos a partir de otros más simples y similares entre sí, gracias a la composición recursiva. El trabajo con el mismo, simplifica el tratamiento de los objetos creados, pues al poseer una interfaz común, todos se tratan de la misma manera.

Percepción: Cualquier transformación de la información en el estado del sistema en una forma conveniente para el uso de un agente.

GLOSARIO DE ABREVIATURAS

CEDIN: Centro de Informática Industrial.

CPU: Unidad Central de Procesamiento.

CU: Caso de uso.

IA: Inteligencia Artificial.

MEF: Máquinas de Estado Finitos.

Ms: Milisegundos.

OO: Orientados a Objetos.

PC: Computadora Personal.

RF: Requisito Funcional.

RnF: Requisitos no Funcional.

RUP: Rational Unified Process o Proceso Unificado de Rational.

RV: Realidad Virtual.

SMA: Sistemas Multi-Agentes.

SP: Sistema de Percepción.

TCG: Tiempo con grupos.

TSG: Tiempo sin grupos.

UCI: Universidad de las Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado.

ANEXOS

Anexo 1: Otras técnicas de comunicación entre agentes

1- Sistemas Basados en Reglas

Los Sistemas Basados en Reglas (SBR) son una técnica de comunicación que trabajan mediante la aplicación de reglas, comparación de resultados y aplicación de las nuevas reglas. También pueden trabajar por inferencia de lógica dirigida, empezando con una evidencia inicial en una determinada situación y dirigiéndose hacia la obtención de una solución, o bien con hipótesis sobre las posibles soluciones y volviendo hacia atrás para encontrar una evidencia existente que apoye una hipótesis en particular.

La programación basada en las reglas es una de las técnicas más comúnmente usadas para crear bases de conocimientos. Las reglas se utilizan para representar conocimiento factual o heurístico, que especifican en un sistema las acciones que se realizarán para una situación dada.

Una regla se compone de una parte "si" y una parte "entonces", donde la parte "si" es una serie de patrones que especifican los hechos (o los datos) que hacen que se aplique una regla. Juntos con los datos, los hechos también están presentes en la base de conocimientos. Estos hechos, representan las declaraciones verdaderas que se utilizan para activar las reglas.

Los SBR tienen dos componentes principales:

a) La memoria activa, que guarda los hechos conocidos y aserciones hechas por las reglas.

b) La memoria de las reglas, que contienen reglas de estilo que operan encima de los hechos guardadas en la memoria activa. Cuando alguna de estas reglas se dispara, pueden activar alguna acción o cambio del estado, como en una Máquina de Estados Finitos, o pueden modificar los volúmenes de la memoria activa agregando nuevas informaciones llamadas aserciones. [12]

2- Lógica Difusa

La Lógica Difusa (LD) es un medio de mostrar los problemas de una manera similar a la forma en que los seres humanos pueden resolverlos y se deriva de la teoría de los conjuntos difusos, su razonamiento se basa en la aproximación a la percepción humana.

La idea de esta técnica es resolver los problemas de una manera imprecisa como lo hacen los seres humanos. Permite plantear y resolver problemas lingüísticos utilizando términos similares a los que un ser humano podría usar.

Esta técnica pertenece a conjuntos cuyos valores varían entre 0 y 1 (ambos incluidos), y permite usar conceptos imprecisos como ligeramente, bastante y mucho. Posibilita también la inclusión parcial en un conjunto.

Todos los términos de la LD pueden ser cierto, pero en diversos grados. Si se dice que determinado aspecto es verdad a grado 1, es absolutamente cierto. Una verdad a grado 0 es absolutamente falsa. Por lo que, se puede tener algo que sea absolutamente cierto, o absolutamente falso; pero también se puede tener algo que esté entre 0 y 1.

Incluye además, el uso de términos lingüísticos intuitivos por ejemplo, cerca, lejos, muy lejos, y así sucesivamente, haciendo que el sistema sea más legible, fácil de entender y mantener.

Esta técnica se puede utilizar para la toma de decisiones así como las aplicaciones. Un ejemplo típico incluye el análisis de la carpeta de valores o de gestión, en virtud del cual, se puede utilizar este sistema para comprar o vender decisiones. La mayoría de los problemas que implica la toma de decisiones sobre la base es subjetiva, vaga o imprecisa. [12]

Anexo 2: Estándares de codificación

Las convenciones o estándares de codificación, son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física. Sirven para facilitar la lectura, comprensión y mantenimiento del código a realizar.

A continuación se presentan algunas de estas convenciones para la programación en lenguaje C++ definidas en la Línea de navegación y comportamiento inteligente del CEDIN.

1. Comentarios

Cada programa deberá comenzar con un comentario (en inglés) que incluya:

- Nombre:
- Descripción:
- Fecha de creación o bitácora de versiones:
- Autor:

Ejemplo:

```
//-------------------------------------------------------------------//

// Name: PSVector3D.h //

// Description: 3D vector struct //

// Version: 1.0 //

// Author: Saily Hurtado Gracia (shurtado@estudiantes.uci.cu) //

//-------------------------------------------------------------------//
```

Cada función debe tener un encabezado que contenga: (Opcional)

- Objetivo de la función y no descripción del procedimiento.

- Comentarios de apoyo a variables, llamadas a función o inclusión de archivos que no sean obvios al proceso.
- Explicación de uso de argumentos (parámetros) no obvios.
- Explicación de uso de valores devueltos (de retorno).

2. Nombres de identificadores

Se considera como identificador a los nombres de variables (arreglos, matrices, apuntadores), funciones, así como cualquier tipo de dato definido por el usuario (estructura, clase). Dichos identificadores deberán seguir las siguientes normas, además de las definidas por el propio lenguaje:

- Deberán tener un nombre significativo para que por su simple lectura, pueda conocerse su función, sin tener que consultar manuales o hacer demasiados comentarios.
- Para nombres que se usen con frecuencia o para términos largos, se recomienda usar abreviaturas estándar para que éstos tengan una longitud razonable. Si se usa abreviaturas, éstas deben manejar la misma lógica en todo el programa.
- Evitar identificadores que comiencen con uno o dos caracteres de subrayado para evitar que se confundan con los que el compilador selecciona.
- Cada identificador de función, variable o procedimiento deberá ser precedido por la abreviación del tipo de dato de que es la variable, o si se trata de una función o procedimiento del tipo de dato que regresa.

3. Identificadores de variables

Comenzarán siempre con la primera letra minúscula correspondiente a su tipo de dato.

Para distinguir palabras dentro del nombre deberá emplearse una letra mayúscula o un guión bajo (_), sin mezclar ambas formas en un mismo programa.

Ejemplos:

- temperaturaDeVapor

- temperatura_de_vapor

Los nombres deberán comenzar con una letra en abreviatura, identificadora de su tipo de dato.

Ejemplo:

Tipo de dato	Abreviatura	Resultado
int	i	iAge
float	f	fPrice
double	d	dPrice
char	c	cTipe
enum	e	eColor

4. Identificadores de funciones

La primera letra deberá ser mayúscula.

Ejemplo:

- void Funcion( );

5. Identificadores de tipos definidos por el usuario

La primera letra será mayúscula.

Ejemplo:

- class Clase o struct Estructura

6. Organización visual del programa

Generales:

- No manejar en los programas más de una instrucción por línea.
- Declarar las variables en líneas separadas.

- Añadir comentarios descriptivos junto a cada declaración de variables, si es necesario.

Sangrías:

- Las sangrías tendrán una longitud de tres espacios.
- Para las llaves que definen el cuerpo de una función, no se deja sangría.

  Ejemplo:

```
bool Funtion()
{
  //Instructions
}
```

- Sangrar las instrucciones del cuerpo de cada estructura de control.

  Ejemplo:

```
bool Funtion()
{
  for (int x = 0; x < 5; x++)
  {
    //Instrucciones a ejecutar
  }
}
```

- Tratar de evitar codificar más de tres niveles de sangrado.

7. Líneas y espacios en blanco

- Las declaraciones de datos dentro de una función, deberán ir al inicio y separadas de las instrucciones ejecutables de la función por medio de una línea en blanco.
- Deben incluirse espacios en ambos lados de los operadores binarios.

  Ejemplo:

```
y = 50 + 15 - x;
```

- Es posible distribuir una instrucción grande sobre varias líneas. Si se hace, seleccionar puntos de ruptura que tengan sentido, como después de una coma en el caso de una lista, o después de un operador en el caso de una expresión larga.

- Sangre todas las líneas subsecuentes.

  Ejemplo:

```
cout << "Ejemplo de ruptura de una instrucción en más de"
     <<" una línea de comandos";
```

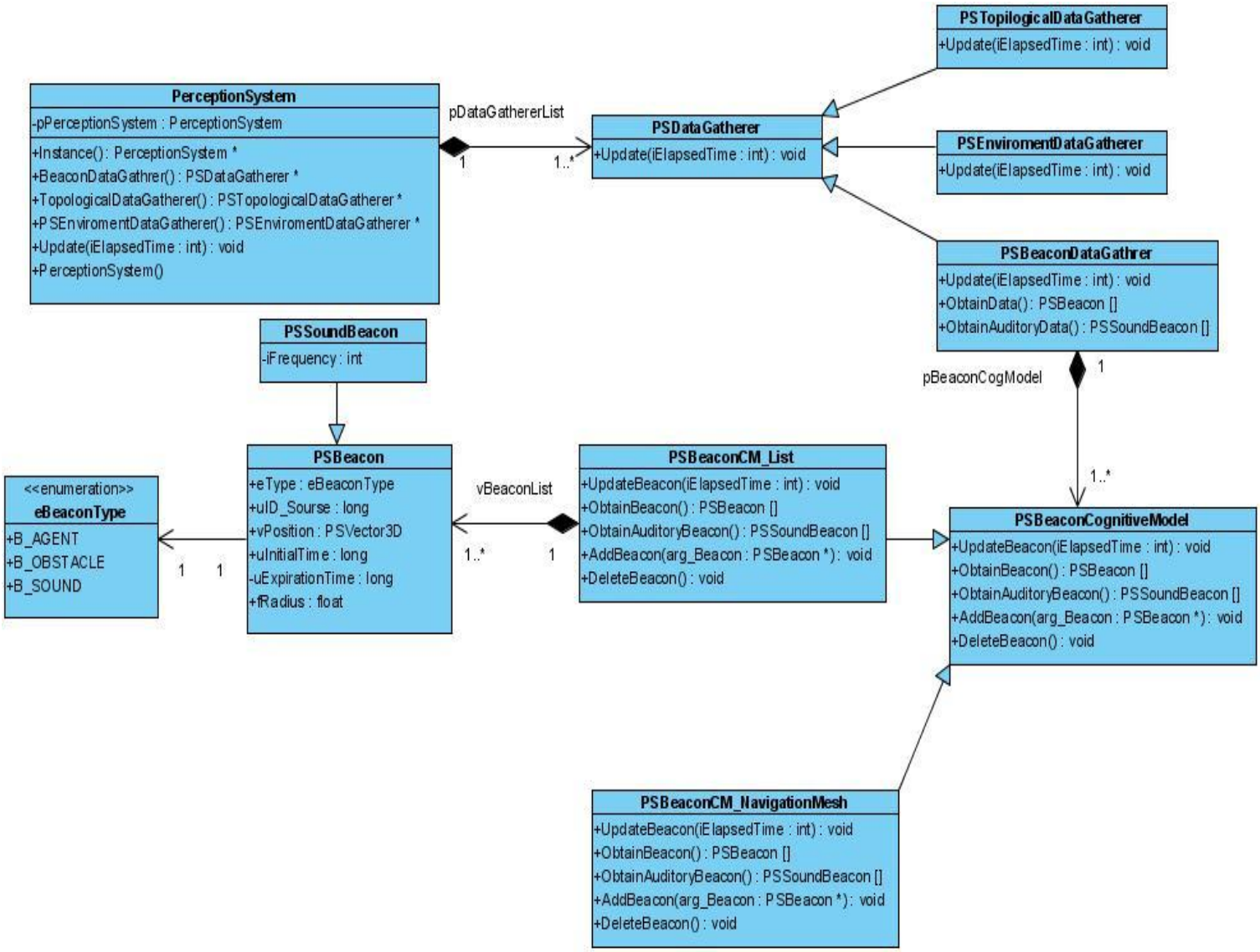
- Los operadores unarios (++, --, etc.) deben ponerse junto a sus operandos, sin espacios intermedios.

- Antes y después de cada estructura de control se deberá poner una línea en blanco.

8. Paréntesis

Para hacer más clara una expresión, es aceptable agregarle paréntesis innecesarios. Dichos paréntesis se llaman paréntesis redundantes.

Anexo 3: Diagrama de clases del diseño. SP



Anexo 4: Diagrama de clases del diseño. Interfaces y sensores