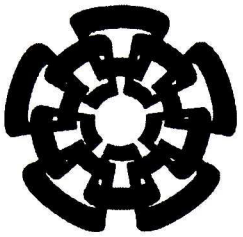
CT-910-551
DON: 7016



Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

# Entorno de desarrollo para arquitecturas cognitivas



Tesis que presenta:
**Armando Cervantes Hernández**

para obtener el grado de:
**Maestro en Ciencias**

en la especialidad de:
**Ingeniería Eléctrica**

Director de Tesis
**Dr. Félix Francisco Ramos Corchado**

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Agosto de 2015.

# Entorno de desarrollo para arquitecturas cognitivas

**Tesis de Maestría en Ciencias**
**Ingeniería Eléctrica**

Por:
**Armando Cervantes Hernández**
Ingeniero en Informática
Universidad Politecnica de Sinaloa 2009-2013

Becario de CONACYT, expediente no. 301006

Director de Tesis
**Dr. Félix Francisco Ramos Corchado**

CINVESTAV del IPN Unidad Guadalajara, Agosto de 2015.

# Índice general

# Índice de figuras

# Índice de figuras

# Índice de tablas

# Resumen

Una arquitectura cognitiva (AC) es una estructura terica con un conjunto de mecanismos para emular el proceso de cognicin humana.

Existen varios proyectos de AC, sin embargo ninguno cuenta con herramientas especificas para el desarrollo de las mismas. Este trabajo se centra en cubrir esas carencias. Proveer un middleware especifico para arquitecturas cognitivas y las herramientas necesarias para mejorar su desempeo y utilizacin.

# Abstract

A Cognitive Architecture is a theroric structure to emulate the human cognitve process.

There exists some AC proyects however no one has specifics tool for the development of them. In this work, we desing a middleware for AC y a set of tool, designed to improve the performance of the middleweare.

# Agradecimientos

Queiro agradecer a mi familia, por el apoyo y ejemplo que han sido para mi a lo largo de toda mi vida.

A mis profesores, por toda la paciencia que han tenido conmigo y por toda la dedicación y empeño con la que dan sus cursos.

A mis amigos y compañeros por todos los momentos vividos, el apoyo y las alegrías.

A CINVESTAV por permitirme ser parte de su comunidad y a CONACYT por brindarme su apoyo económico.

Y finalmente a Flor, por cada momento vivido en este tiempo, los hermoso, los graciosos, los difíciles, cada uno de ellos es un valioso recuerdo para mi. Un logro mas en el que estamos juntos.

Muchas gracias a todos.

# Glosario

- Entidad: En este documento la palabra entidad es manejada como sinónimo de entidad virtual. Una criatura que habita en un ambiente virtual y exhibe un comportamiento.
- Middleware: Es una capa de abstracción de software, la cual provee comunicación transparente entre dos componentes de un sistema.
- Modulo: Componente aislado de un sistema distribuido. Pieza independiente que forma parte de un sistema.
- BigNode: Los BigNode's son los módulos principales del middleware, se encargan de la localización y comunicación de todos los mensajes a través del sistema. Estos se coordinan para poder hacer una efectiva actualización de tablas de ruteo.
- SmallNode: Es un módulo que se encarga de ejecutar un trabajo y enviar los resultados.
- Multicast: Es un protocolo de comunicación en el cual el router crea una lista de subscriptores bajo petición y el paquete que llegue al grupo será reenviado a todos los subscriptores.

# Capítulo 1

# Introducción

En este capítulo vamos a analizar cuáles son las necesidades que llevaron a la realización de esta investigación, analizaremos cuales son los conceptos básicos para leer esta tesis, asi como también plantearemos los problemas que este trabajo intenta resolver y el contexto en el que estos se desarrollan. También daremos una muy pequeña descripción de la propuestas.

Una arquitectura cognitiva (AC) es una estructura teórica con un conjunto de mecanismos para emular el proceso de cognición humana [2]. Las ACs proporcionan la infraestructura necesaria para un sistema inteligente.

El objetivo inicial de la inteligencia artificial era crear una entidad capas de exhibir de un comportamiento catalogado como inteligente.

Sin embargo con el pasar del tiempo, se a enfocado a crear técnicas y algoritmos para resolver problemas específicos.

Una arquitectura cognitiva típicamente incluye los siguientes elementos [3] [4] :

- Memoria de corto y largo término que almacenan el contenido de las creencias, metas y conocimiento del agente.
- La representación de elementos contenidos en la memoria y su organización a través de estructuras mentales de gran escala.
- Procesos funcionales que utilizan esas estructuras, incluyendo mecanismos de desempeño y mecanismos de aprendizaje que modifiquen estas memorias.

La mayoría de las arquitecturas cognitivas están basadas en componentes muy específicos, enviando información entre ellos a través de un flujo de datos. La creación de arquitecturas cognitivas que representan el comportamiento de áreas del cerebro pueden ser abstraídas como sistemas distribuidos, esto conlleva a una serie de retos de implementación propia de estos sistemas.

## 1.1. Descripción del problema

El grupo de investigación al que pertenezco esta desarrollando una arquitectura cognitiva basado en evidencia neurocientífica. Dicha arquitectura posee varias funciones cognitivas: Memoria, percepción, atención, emoción, sistema motor, motivación, planeación y toma de decisiones.

- Memoria.- Se encarga de almacenar la información a corto y largo plazo. Debe de establecer el formato de toda esta información y sus interrelaciones.
- Sistema sensorial.- Se encarga de obtener directamente las entradas del ambiente y sus cambios a través del tiempo.
- Atención.- Involucra mecanismos para filtrar la información irrelevante, y de esa manera seleccionar la importante..
- Emociones.- ...

  Sistema motor.- Se encarga de enviar señales a los diferentes músculos del sistema y coordinarlos en sus funciones.
- Motivación ...
- Planearon.-
- Toma de decisiones.

Cada función cognitiva esta siendo desarrollada por uno o dos investigadores. Dando como resultado un complejo sistema que requiere correr múltiples módulos independientes y por cuestiones de factibilidad (recursos insuficientes para un solo equipo), esto se tiene que correr a través de un sistema distribuido.

El equipo requiere una herramienta para integrar todo el conocimiento y resultados generado por el grupo. Un medio común donde se pueda converger a través de una implementación estandarizada entre todas las funciones cognitivas.

Este trabajo se centra en el desarrollo de un conjunto de herramientas que solventen los problemas de abstracción, implementación, comunicación, integración y estandarización a los que el grupo de investigación se enfrenta al momento de pasar de los modelos cognitivos a módulos de un software.

## 1.2. Justificación

En la actualidad existen middleware's de propósito general ( como CORBA [5]) y pocos middleware enfocados en arquitecturas cognitivas (como LIDA [6] y YARP [7]), sin embargo cada uno cuenta con sus limitaciones y sus diferentes enfoques.

Otro tema es la nula existencia de entornos de desarrollo integrado que permitan de manera intuitiva la implementación de modelos cognitivos.

Si no se contara con esta herramienta, el equipo de investigación carecería de:

- Un consenso global de que tecnología usar para resolver el problema de comunicación entre los diferentes módulos.
- Un estandar para adaptar los modelos cognitivos a las diferentes limitaciones que una tecnología imponga al equipo de investigación.
- Capacidad para detectar intersecciones entre funcionalidades y diversos conflictos entre sus implementaciones.
- Un plan a largo plazo para la integración entre las intersecciones de todos los módulos del sistema.

## 1.3. Objetivos

### 1.3.1. Generales

Los objetivos generales de este trabajo son:

- Proveer un software que proporcione una solución al problema de comunicación y locación de los módulos del sistema.
- Asegurarse que este software cumpla con los requerimientos del grupo de arquitecturas cognitivas (estándar planteado en [8].
- Desarrollar un Entorno de Desarrollo Integrado, que proporcione herramientas necesarias para la abstracción de modelos cognitivos.

### 1.3.2. Específicos

- Llegar a un estándar de software para el grupo de arquitecturas cognitivas, satisfagan todas las necesidades del estándar de modelado.
- Transladar los requerimientos de usuario en requerimientos de sistema
- Investigar si existen alternativas para los requerimientos
- Proponer la arquitectura de middleware que mejor cumpla con los requerimientos

  Implementar el middleware

  Validar que el middleware cumpla con los requerimientos de sistema

  Levantar requerimientos para un IDE
- Investigar alternativas que puedan cumplir con estos requerimientos
- Implementar el IDE
- Validar que el IDE cumpla con los requerimientos deseados

## 1.4. Propuesta

Nuestra solución propuesta es un middleware acompañado de un entorno de desarrollo integrado para poder resolver el problema de unificación y abstracción de modelos cognitivos.

En la figura 1.1, se observa un middleware como una serie de nodos interconectados, ademas de una capa de software para que la utilización del middleware sea mas sencilla.

## 1.5. Estructura del documento

Este documento esta estructurado de una forma sencilla, que lleva al lector a través de los diferentes proceso envueltos en el desarrollo de un software. Los capítulos incluidos son:

- Marco Teorico: En esa sección examinamos el conocimiento que se tiene en el área en la que se desenvuelve este trabajo, ofreciendo un breve repaso en el área de ACs y su categorizacion. Posteriormente examinamos el estado del arte de los middleware, que son la altamente requeridos en la implementación de cualquier AC. Y también se da la justificación del porque la necesidad de este trabajo.

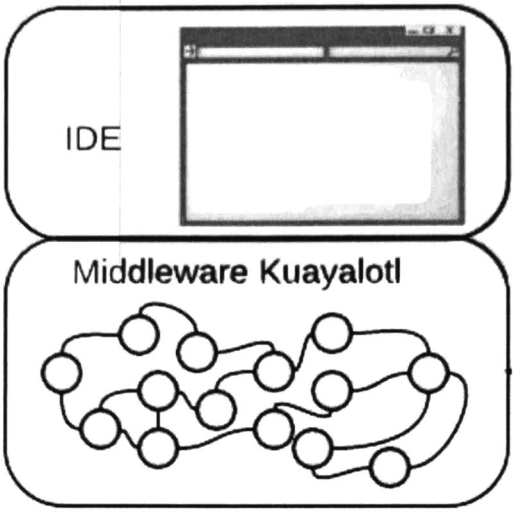



Figura 1.1: Estructura basica de la propuesta: Un middleware con un Entorno de desarrollo integrado que funcione como capa de abstracción

- Propuesta: En esta capitulo exploramos el proceso de planeación e implementación de nuestra propuesta, presentando los puntos mas trascendentes de la misma. Así como también un trabajo derivado de la propuesta; un plugin para la incorporación en un Entorno de desarrollo integrado, el cual complementa la herramienta que se presenta.
- Resultados: Aquí se hace un análisis del desempeño del middleware, presentando los casos de ejemplo que se utilizaron para ello. Así como también un caso de estudio que se realizo utilizando la herramienta y el plugin para el entorno de desarrollo.

# Capítulo 2

# Marco Teórico

En esta sección se analiza a profundidad los conceptos de una arquitectura cognitiva, cuáles son sus componentes, fundamentos, objetivo. También se analizara los tipos de arquitecturas cognitivas que existen, además de sus ventajas y desventajas. Posteriormente se analizará el tema de los middleware, los tipos que existen, las tecnologías que manejan y sus diversas ventajas y desventajas.

## 2.1. Arquitecuras Cognitivas

Como ya se menciono, una arquitectura cognitiva (AC) es una estructura teórica con un conjunto de mecanicismos para emular el proceso de cognición humana [2]. Las arquitecturas cognitivas se basan en las ciencias cognitivas para presentar fundamentos teóricos del proceso que se trata de emular. Principalmente se utilizan las áreas de neurociencias o psicología. Una arquitectura cognitiva típicamente incluye módulos de memoria a corto y largo plazo una representación de los elementos de la memoria y procesos que utilizan y alimentan estas estructuras de memoria (toma de decisiones, plantación, percepción etc.) [4] [3]. En la figura 2.1 se puede observar la relación típica de estos componentes y su ambiente.

Dejando de lado los componentes típicos de las arquitecturas cognitivas están también se pueden catalogar por la manera en la que almacenan la información. Existe una fuerte relación entre el tipo de arquitectura que se utiliza y el problema a resolver por esta. Aunque existen varias subdivisiones de cada una, las mas destacadas son: simbólica, Emergente e híbrida. Sus principales características se pueden apreciar en la figura 2.2.

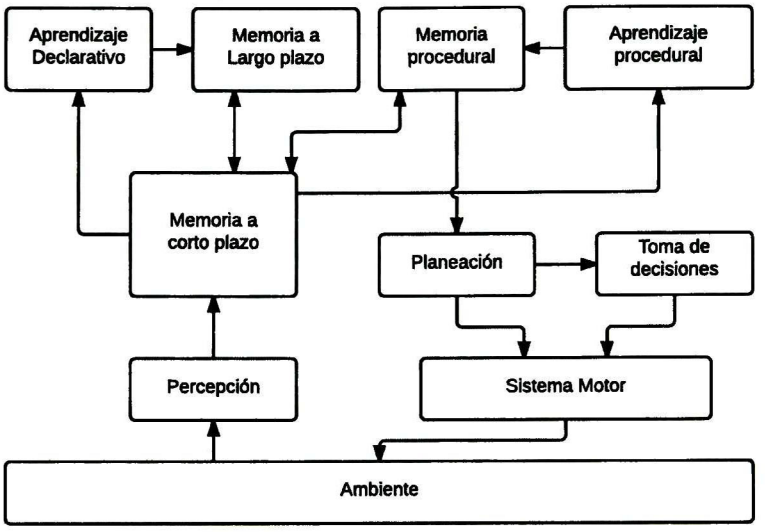



Figura 2.1: Estructura típica de una arquitectura cognitiva - basada en el diagrama de Laird [9]

### 2.1.1. Simbólico

La aproximación simbólica almacena la información principalmente como símbolos de un lenguaje de primer orden [10]. La información del ambiente es percibida, abstraída y entonces representada como símbolos. El proceso de aprendizaje permite que se agreguen nuevos símbolos y reglas. Esto nos brinda la oportunidad de utilizar las entradas y procesos de una manera no ambigua permitiendo reaccionar de manera apropiada para alcanzar la meta del sistema. La mayoría de las arquitecturas que trabajan con la aproximación simbólica procesan esta información de una manera centralizada a través de un flujo serial de datos. La manera típica en que estos sistemas representan la información es mediante un grafo dirigido siendo los nodos las entidades y sus atributos y los enlaces representan sus diferentes relaciones[11]. Los principales ejemplos de esta representación del conocimiento son las redes semánticas y los grafos conceptuales[12].

### 2.1.2. Emergente

En un sistema emergente la cognición es el procesó de adaptación al ambiente. Estos sistemas pasan por un proceso de auto-organización para responder al ambiente y mantenerse operable.

La principal inspiración de los sistemas emergentes vienen de las ideas conexionistas [13].

Se basa principalmente en elementos de procesamiento que forma redes de nodos interconectados de maneras específicas y cambiando sus estados internos para en conjunto generar un comportamiento emergente. La memoria se puede representar de manera local o de manera global. Los métodos de aprendizaje de las arquitecturas emergentes suelen ser muy variados [13]. Aunque existen dos tendencias principales en los métodos de aprendizaje, asociativo y competitivo.

- Aprendizaje asociativo.- Crea relaciones de entradas específicas con salidas específicas. Se usan técnicas como identificación de patrones. El problema en estos casos se origina porque es necesario contar con un conjunto correcto de señales previamente y de ese modo asociar las que sean similares.
- Aprendizaje competitivo.- En este paradigma de aprendizaje, todos los elementos de procesamientos compiten para poder ser activados y aprender de las señales activadas. Ejemplo: Un conjunto de módulos recibe un patrón y estos envían una respuesta, la respuesta mejor evaluada pasa por un proceso de aprendizaje para ser refinada aún más.

### 2.1.3. Híbridos

Debido a que los paradigmas simbólicos y emergentes tienen sus ventajas y desventajas, también existen tendencias que intentan combinar ciertos conceptos de cada paradigma para lograr un marco de trabajo mucho mas flexible [14]. Las arquitecturas simbólicas son capaces de realizar funciones cognitivas de alto nivel con una buena abstracción. Pero se complica al momento de tratar de usar reglas simbólicas para dar resultados con información ambigua. Por otra parte, las arquitecturas emergentes son adecuadas para manejar información mas simple con un enfoque de paralelismo. Sin embargo esto se dificulta al momento de realizar tareas cognitivas de alto nivel. Las investigaciones en esta área son muy diversas. Sin embargo se podría dividir en dos categorías principales: local-distribuida y simbólica-conexionista [14].

- Local-distribuida.- En esta tendencia se hace una combinación de como el se deben realizar las técnicas de aprendizaje. Permitiendo tanto que existan conceptos que son

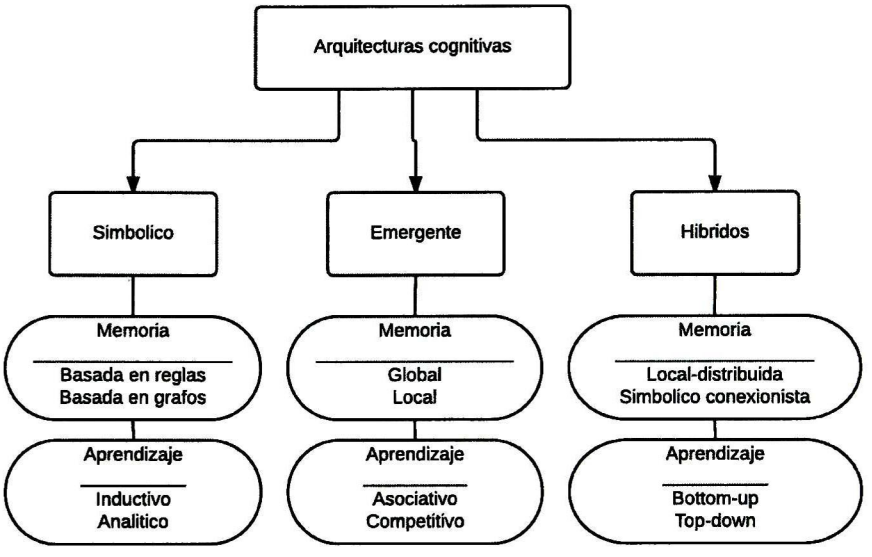



Figura 2.2: Categorías de las arquitecturas cognitivas - basada en el diagrama de DUCH [11].

aprendidos por un solo nodo y conceptos que son reconocidos por un conjunto de nodos [11].

- Simbólica-conexionista.- En esta tendencia se combinan los dos métodos de almacenar la información en la memoria. Existen módulos que almacenan la información de manera simbólica (basada en reglas o en grafos) así como también módulos conexionistas(local o distribuida).

Cada tipo de arquitectura cognitiva tiene sus ventajas y desventajas, algunas son limitas por la rigidez, otras por el caudal de información que necesitan manejar y el poco nivel de abstracción en sus procesos. Sin embargo cada una tiene el mismo objetivo final: Conseguir una inteligencia artificial genérica, la cual pueda adaptarse automáticamente para poder resolver problemas para los que no fue programada específicamente.

## 2.2. Middleware

Un middleware es una capa de abstracción que ofrece una manera de comunicar software a través de una red de dispositivos, siendo indiferente ante hardware o plataforma [15]. Esta capa de abstracción cuenta con una interfaz de programación de aplicaciones (API), que provee un conjunto de métodos para poder comunicar las implementaciones corriendo sobre el middleware de una manera transparente para el usuario.

Principalmente un middleware provee capas de abstracción para la heterogeneidad del software y el hardware. Algunos middleware también proveen heterogeneidad para el sistema operativo o el lenguaje de programación. Además de transparencia para uno o varios de las siguientes dimensiones: Locación, concurrencia, replicación y movilidad.

Estos problemas son solventados a través de servicios, que forman un conjunto de funciones y protocolos estandarizados para solventar problemas comunes (como identificar el tipo de plataforma en la que se ejecuta, encontrar un modulo especifico dentro de la red, etc..) la utilización de estos servicios tiene que ser transparente para el usuario, a su vez el usuario tiene que poder acceder a todas las funciones que los servicios brindan a través de la API.[16].

De acuerdo a los metodos de comunicacion que utilizan, los middlewares se dividen en diferentes tipos de categorias [17]:

- Llamada a procedimiento remoto.

  Objetos Distribuidos.

  Orientado a mensajes.

- Tuplas distribuidas.

- Transaction Processing Monitors

  Servicios Web

A continuación esta una descripción detallada de cada uno de ellos, junto con sus ventajas y desventajas.

### 2.2.1. Llamada a procedimiento remoto

Una llamada a procedimiento remoto (RPC) consiste en extender la abstracción habitual de las llamadas a procedimiento, al paradigma de los ambientes distribuidos, permitiendo llamar un procedimiento que se puede encontrar o no en el equipo, como si fuera local de manera

transparente para el usuario[15]. Los middleware que ofrecen este tipo de paradigma proveen una capa de abstracción que enmascara la locación e incluso el lenguaje de programación.

El estándar mas conocido que existe es el Llamado a procedimiento remoto de Ambiente de computo distribuido (DCE RPC) de la Open Software Fundation. Una implementación de este estándar debe de incluir un compilador que convierta una interfase en un método del cliente, el cual transforma los parámetros en un paquete. Además debe de generar un método server, el cual obtiene y procesa el paquete.

Estos métodos pueden enviar información interoperable entre diferentes lenguajes de programación. En la figura 2.3 se puede observar la arquitectura de la implementación de este estándar.

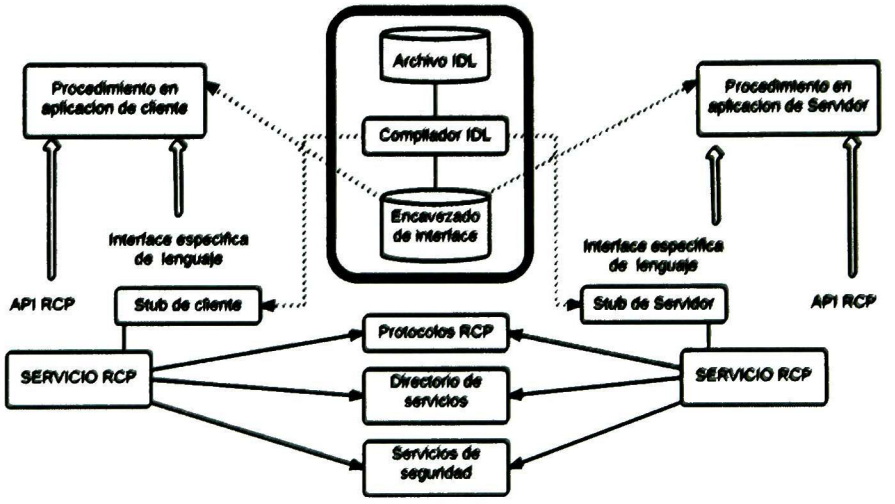



Figura 2.3: Arquitectura de la implementación del estándar DCE RPC basada en el diagrama de Bernstein[16].

### 2.2.2. Objetos Distribuidos

El paradigma de objetos distribuidos supuso un gran cambio en su tiempo ya que provee un método de comunicación no ambiguo y casi transparente de comunicación. Este paradigma esta basado en el paradigma RPC, sin embargo la abstracción a objetos permitió una manera mucho mas sencilla de representar las conexiones entre los módulos. Este enfoque provee una manera de construir software entre elementos bien definidos y casi independientes. [18]

Un middleware para objetos distribuidos debe de proveer mecánicas para poder crear las interfaces de los métodos, para que estos sean visibles desde otras locaciones, permitiendo adoptar el paradigma orientado a objetos en los sistemas distribuidos, se usa como una herramienta para poder esconder un poco de la complejidad de los mismos. [17][15]

Ejemplos de middleware de este tipo son:

#### 2.2.2.1. CORBA

CORBA [5, 19] es un middleware de propósito general diseñado para resolver problemas recurrentes de sistemas distribuidos, tales como transparencia de locación, interacción entre lenguajes, descriptores de interfaces y acceso concurrente. Además de traer las ventajas del paradigma orientado a objetos y hacerlo útil para los sistemas distribuidos.[15]

CORBA trabaja con la arquitectura Negociante de solicitudes de objeto (ORB) (figura 2.4), la cual es un mecanismo por el cual un cliente se puede comunicar e invocar métodos en objetos que pueden estar en en diferentes locaciones y en diferente lenguaje en una manera completamente transparente para el usuario [20].

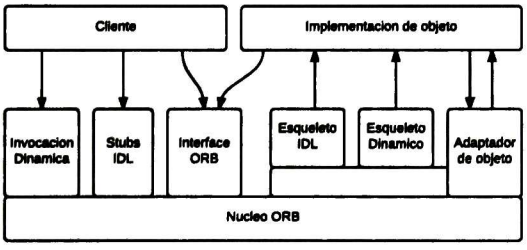



Figura 2.4: Diagrama de la arquitectura de CORBA ORB

CORBA provee un lenguaje de descripción de internase (IDL) para describir las entradas y salidas de cada método, creando una manera no ambigua de comunicarse entre diferentes objetos. Este IDL se tiene que compilar para poder generar stubs. Estos se tienen que generar por cada cliente y están escritos en el mismo lenguaje del cliente. El ORB utiliza estos stubs para poder empaquetar y desempaquetar los mensajes de una manera transparente y no ambigua para el cliente y el servidor.

La principal ventaja de CORBA es poder distribuir trabajo Secuencial en un paradigma distribuido sin que el usuario se percate de ello. No obstante, el enfoque de el equipo de investigación al cual pertenezco, es crear una gran red de módulos pequeños para distribuir el trabajo, cada módulo funciona de forma independiente a los otros, haciendo que las ventajas de CORBA nos sean relevantes para nuestro enfoque.

#### 2.2.2.2. BALT & CAST

BALT & CAST [21] es una herramienta diseñada para resolver problemas recurrentes en la implementación de robots cognitivos. Sigue el esquema arquitectural Cosy (CAS)[22], el cual esta basado en la intercomunicación de sub-arquitecturas.

Cada sub-arquitectura tiene una memoria de trabajo individual y un manejador de tareas. BALT & CAST trabaja con una capa de abstracción construido encima de CORBA y cuenta con un software llamado CAS Toolkit, que provee clases abstractas para crear componentes claves de un CAS. Sin embargo, BALT & CAST es escasamente documentado y cuenta con una arquitectura muy restrictiva, ya que obliga a los objetos a actualizarse por medio de lecturas de un buffer.

### 2.2.3. Orientado Mensajes

A medida que los sistemas aumentan de escala, el flujo de datos y una disponibilidad necesaria de 24 horas,los sistemas RPC comienzan a ser insuficientes Por este motivos se ideo un sistema que permitiera una tasa mayor de concurrencia, permitiendo que las peticiones pudieran ser manejadas a través de colas.

Los Middleware Orientados a Mensajes (MOM) proveen un mecanismo de comunicación sencillo entre diferentes módulos o entidades de un sistema. Un MOM puede ser definido como cualquier infraestructura de middleware que posea capacidades para enviar y recibir mensajes entre módulos independientes[23].

Un elemento MOM puede enviar y recibir mensajes con otros elementos MOM. La comunicación entre estos elementos es par a par (p2p), esto significa que dos equipos se envían mensajes directamente, sin intermediarios que estén reenviando los mensajes. Esto permite una mayor flexibilidad al sistema, permitiendo implementar cambios en ciertos módulos sin tener que afectar a los demás.

Existen dos tipos de MOM, estos se diferencian en la forma en la que procesan las solicitudes, los tipos son: Síncrono y Asíncrono.

#### 2.2.3.1. Comunicación Síncrona

Como se puede observar en la figura 2.5, la comunicación síncrona, una vez que un módulo envía un mensaje este es bloqueado y debe esperar una respuesta para continuar con su ejecución. Esto puede conllevar a que se generen puntos muertos (deadlocks), que el sistema se quede esperando una respuesta de manera permanente y por lo tanto colapse, imponiendo restricciones similares a los sistemas RPC [23].

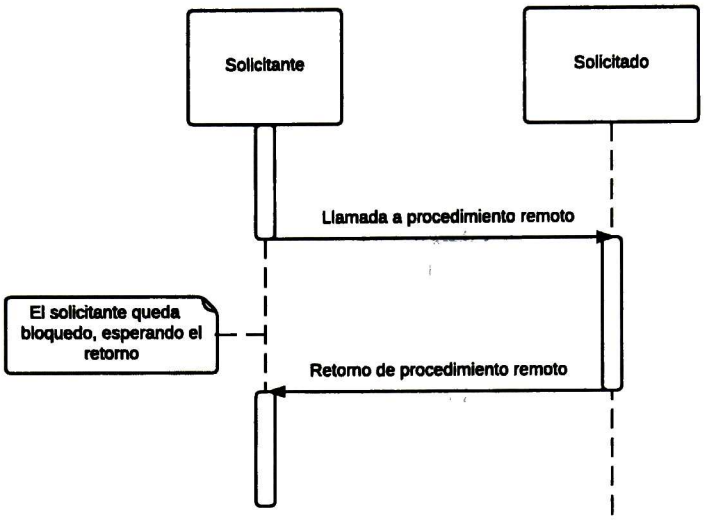



Figura 2.5: Procedimiento de el envío de un mensaje en comunicaciones síncronas - basada en el diagrama de Curry [23].

#### 2.2.3.2. Comunicación Asíncrona

En el modelo asíncrono, el modulo que envía el mensaje continua la ejecución independientemente si el mensaje obtuvo una respuesta.

En este modelo se ocupa normalmente un método para poder retener tareas pendientes, normalmente este método es una cola de mensajes [23].

Algunos ejemplos de este tipo de MOM síncronos son:

#### 2.2.3.3. Yet Another Robot Platform

*Yet Another Robot Platform (YARP)* [7] es un middleware desarrollado con el propósito de facilitar a los investigadores la implementación de sus algoritmos en tiempo real, sin preocuparse por cuestiones de infraestructura. La arquitectura congitiva iCub[24], usa YARP para su implementación.

YARP corre sobre un cluster de computadoras conectadas a través de la red y distribuye el trabajo sobre ellas. En este cluster, el código debe ser dividido en módulos, para poder proveer una mejor capacidad de reutilización y mantenimiento. Estos módulos tienen la propiedad de

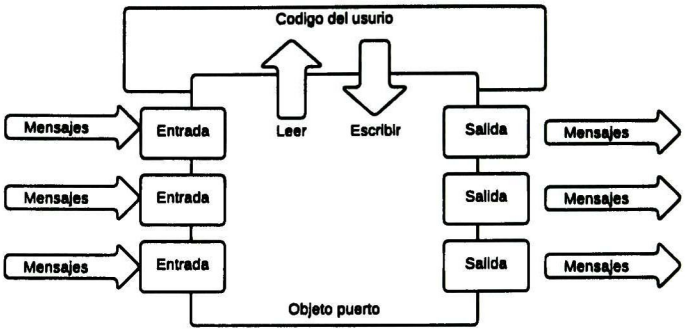



Figura 2.6: Estructura básica de un elemento de YARP - basada en el diagrama de [7]

ser independientes de locación y de comportamiento entre ellos (la ejecución de un modulo no interfiere con los demás).

YARP trabaja con el patrón Observador [25], la base del patrón observador es un conjunto de objetos puertos(OP). Un OP se encarga de enviar y recibir los mensajes de cada modulo. Un OP puede observar un conjunto de otros OP(Entradas). Un OP a su vez puede ser visto por otro conjunto de OP (Salidas).

Desafortunadamente el enfoque de YARP es el proveer poder de procesamiento para algoritmos de reconocimiento y de visión, además de que los procesos de descubrimiento de módulos en YARP, son controlados por un servidor denominado *"Name server"* [7](El cual tiene que ser ingresado manualmente), ligando el middleware a una entidad centralizada y presentando un cuello de botella para el sistema.

#### 2.2.3.4. BAVOI

En Brain Architecture for Visual Object Identification [26] se propone un MOM basado en módulos interconectados que representan áreas del cerebro y cada módulo es implementado como un módulo ejecutándose de forma transparente en una red de computadoras. Las principales características es que cada modulo que representa un área del cerebro se puede comunicar libremente con cualquier otro.

En la figura 2.7 podemos observar la arquitectura del sistema. Se nota la naturaleza jerárquica del mismo. Los componentes principales son:

- Main Middleware.- Componente que se encarga de almacenar todas la direcciones de los demás componentes del sistema.

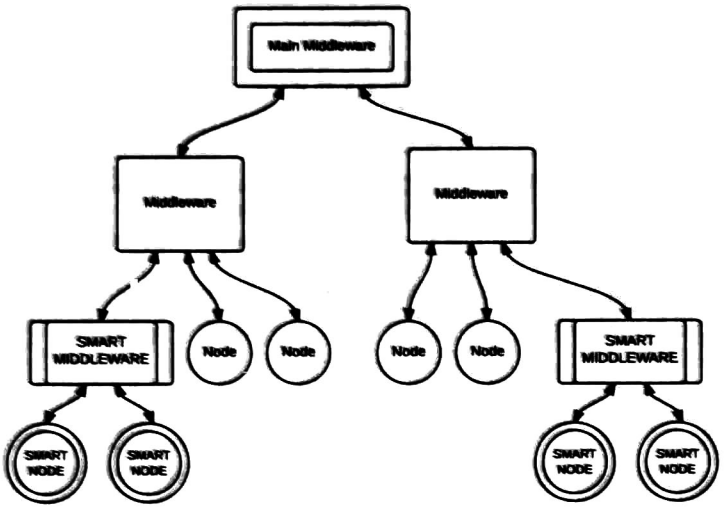



Figura 2.7: Arquitectura del middleware Bavoi - Basado en la descripción de [26].

- Middleware.- Se encarga de administrar los nodes y los smart Middleware(descritos mas abajo), así como de solicitar información al Main Middleware.
- Node.- Unidad de procesamiento básico, aquí es donde los paquetes son destinados para ser procesados.
- Smart Middleware.- Es un componente que puede crear muchas instancias de la misma clase Smart Node, y de esta manera poder paralelizar el trabajo.
- Smart Node.- Es un modulo Node que puede ser instanciado varias veces.

Como se observo en la figura 2.7 el sistema posee un modulo central llamado "Main Middleware", este puede representar un cuello de botella muy importante cuando la escala del sistema aumente.

#### 2.2.3.5. JADE

Java Agent Development (Jade) [27] es un MOM y un framework completamente implementado en Java. JADE provee un middleware para desarrollar sistemas multiagentes. Trabaja a través de un sistema de páginas blancas y amarillas [28] para integración de servicios. Además de un mecanismo de descubrimiento para nuevos componentes.

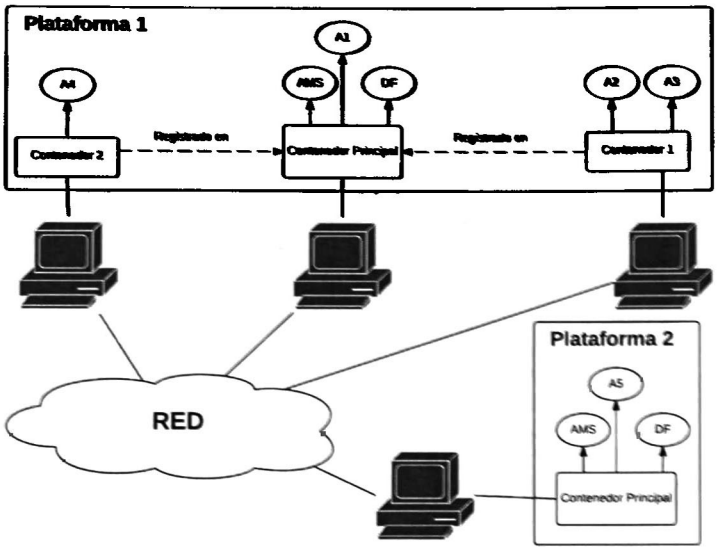



Figura 2.8: Arquitectura de JADE.

JADE cumple con la especificación de la *fundación para agentes físicos inteligentes*(FIPA)[29] Cuenta con un conjunto de herramientas visuales que soportan el proceso de depuración y almacenamiento de resultados. JADE es multiplataforma debido a su implementación en JAVA.

En la arquitectura de JADE (figura 2.8) se puede observar que cada computadora que ejecuta JADE cuenta con un componente llamado çontenedor", el cual se encarga de proveer de la comunicación de los agentes dentro del sistema.

El sistema concentra su funcionamiento en el çontenedor principal", el cual incluye dos agentes, el agente AMS, el cual es el único capaz de manipular el sistema completo (iniciar nuevos agentes, detener el sistema etc...) y el agente DF; el sistema de paginas amarillas el cual puede publicar y encontrar servicios para otros agentes.

Esta misma arquitectura hace que el procesamiento en JADE se acumule en el çontenedor principal" limitando la capacidad del sistema y presentando un cuello de botella para el desempeño, además las herramientas que JADE provee están orientadas al monitoreo de comunicación en simulación de sistemas multiagentes.

### 2.2.4. Tuplas distribuidas

Una tupla es un arreglo de datos, al ser distribuida, esto significa que varios procesos pueden acceder a estos datos de forma simultánea [30].

Una base de datos relacional distribuida puede ser considerada como una abstracción de tuplas distribuidas, esto puede considerarse el modelo mas extendido de middleware de la actualidad. La mayoría de estos sistemas ofrecen acceso a través de un lenguaje de consulta estructurado (SQL), esto provee una capa de abstracción tan intuitiva como rigurosa.

La mayoría de estos sistemas implementan de manera nativa abstracciones para procesos como transacciones, heterogeneidad a través de diversos lenguajes y plataformas[17].

#### 2.2.4.1. Linda

Linda es un framework que ofrece una abstracción de tuplas distribuidas llamadas Espacio de tuplas (*Tuple Space*). Las características de Lida incluyen abstracción espacial, ya que los clientes pueden acceder al Tuple Space sin importar su ubicación y abstracción temporal ya que evita que consultas sobre la Tuple Space pueden sobreponerse en la interacción con un objeto.

La estructura de Linda consiste en un pequeño conjunto de procesos de comunicación y de control, que apoyan la creación y manipulación de estructuras de datos distribuidas.[30].

### 2.2.5. Monitores de procesamiento de transacciones

Una transacción es una unidad de trabajo que debe de ser ejecutada una vez y debe de presentar un cambio permanente. Esta tiene tres propiedades [31]:

- Serializable: Las transacciones tienen que llevarse a cabo de manera serializada.
- Todo o nada: Si una transacción sufre un error en su ejecución, tiene que deshacer los efectos que hasta ese punto se hicieron.
- Persistente: Los cambios realizados por una transacción deben de ser permanentes.

Un sistema de monitoreo de transacciones se asegura de que todas las transacciones de los usuarios de un sistema, sean ejecutadas de manera coordinada y correcta en secuencia y forma. Este tipo de sistemas típicamente cuenta con los siguientes módulos [31]:

- Un manejador de mensaje: se encarga de recibir, formatear y validar los mensajes de todas las terminales conectadas al sistema.

- Controlador de peticiones: se encarga de administrar las peticiones que vienen en los mensajes y de ordenarlas adecuadamente.
- Servidor de aplicación: Es el programa que accede directamente a la base de datos para aplicar los cambios.

### 2.2.6. Servicios Web

Servicios WEB en el estricto sentido de la definición no es un middleware, sin embargo si es una serie de estándares y paradigmas que proveen una capa de abstracción para la comunicación.

Existen muchas definiciones muy variadas para los servicios web, pero una de las mas sencillas lo define como una aplicación accesible a otras aplicaciones a través de la web [32].

Una de las características principales de los servicios web, la que le permite un grado enorme de flexibilidad, apertura y de combatividad pero a la vez incrementa considerablemente el peso de los mensajes, es que los mensajes están codificados en XML como texto plano. De esta manera cada lenguaje puede interpretar el mensaje como una cadena de caracteres auntodescriptiva y así realizar una conversión de acuerdo a sus necesidades.

Los servicios WEB han sido criticados muy fuertemente por su falta de interfaces, permitiendo que se envié casi cualquier mensaje entre los módulos, ademas de que el peso de los mensajes es enorme comparado con otros estándares.

# Capítulo 3

# Propuesta

En este capítulo presentamos nuestra propuesta para solucionar el problema planteado. La propuesta esta compuesta básicamente de dos componentes. Un middleware que es presentado en la sección 3.1 de este capítulo. y un conjunto de herramientas que facilitan la implementación de una arquitectura cognitiva al través de un entorno de desarrollo gráfico y un lenguaje de descripción que son descritos en la sección 3.2 de este capítulo.

## 3.1. Middleware

El problema que resuelve el middleware es la integración de funciones cognitivas desarrolladas por diferentes miembros del equipo que necesariamente no adoptaron convenciones para el desarrollo de dichas funciones. Es decir, el middleware implementa una tecnología común, para que las implementaciones sean compatibles, solventa problemas de heterogeneidad tanto de infraestructura como de software (diferentes tipos de plataformas, lenguajes de programación, protocolos de comunicación, etc.) ya que no son el principal problema de interés para la investigación de ACs.

Previamente el grupo de investigación ya contaba con un middleware descrito en [33]. Sin embargo dicho desarrollo no cumplía con los requerimientos del equipo. Para facilitar continuidad en el uso de este middleware, se continua utilizando los componentes principales de la propuesta inicial (big node, small node, eference, etc. [33]). A continuación se presenta el proceso que se siguio para desarrollar el nuevo middleware.

### 3.1.1. Requerimientos del Sistema

Para hacer el levantamiento de requerimientos se reunió el equipo de investigación donde se discutieron las necesidades individuales y en conjunto. Esta tarea resulto compleja, por lo que finalmente se decidio generar una página donde cada miembro del equipo planteaban sus requerimientos y estos eran visibles a todos los miembros del equipo de investigación. Despues de una serie de juntas se concluyo con los requerimientos de usuario, los cuales fueron trasladados a diversos requerimientos de sistema. A continuación se muestran algunos como ejemplo y en el apendice A se muestran todos los requerimientos descritos. Es necesario resaltar que estos requerimientos fueron levantados empleando el estandar IEEE830:

| ID | requerimiento |
|---|---|
| RS004 | El middleware permitirá de manera transparente la ejecución distribuida en un ambiente local. |
| RS006 | Pueden correr varios Big Nodes (BN) en el mismo equipo. |
| RS007 | La intersección de funcionalidades de BN con el mismo nombre debe de ser transparente. |
| RS008 | La función de los BN debe de ser encaminar la información a los Small Nodes(SN). |
| RS009 | Los BN obtendrán las direcciones de otros BN conforme se van registrando. |
| RS010 | El registro de los BN se debe realizar por multicast. |
| RS011 | La solicitud de dirección de otro BN se debe realizar por multicast. |
| RS013 | Un mismo BN podrá manejar varios tipos de SN. |
| RS014 | Un SN podrá enviar información directamente a otro SN del mismo SN. |
| RS018 | Los nodos deben de poder configurarse como persistentes o momentáneos. |
| RS021 | Debe existir una jerarquía de BN, la cual permitirá tener niveles en el ruteo de los paquetes. |
| RS022 | Cuando un BN sea inicializado este enviara un mensaje multicast, para informar de su existencia. Este mensaje debe de ser contestado por los demás BN enviando sus datos de ruteo. |
| RS025 | El ID incluirá los campos para identificar BN, tipo de SN, SN. |
| RS026 | El ID estara dividido en: Area: 12 bits, tipo: 5 bits, Index: 15 bits |
| RS034 | Cuando un BN ingrese al grupo multicast, mandara un mensaje SINGINAREA, el cual al contendrá el ID, IP y Puerto del BN que ingresa. |
| RS035 | Cuando un BN reciba un SINGINAREA, este agregara al nodo entrante a su tabla de ruteo y contestara con un mensaje HANDSHAKE, el cual contendrá el ID, IP y Puerto del nodo que recibió el SINGINAREA. |
| RS036 | Al recibir un HANDSHAKE el nodo agrega el emisor a su tabla de ruteo y envía los mensajes de DATA pendientes en caso de existir. |

Continua en la siguiente página

| ID | requerimiento |
|---|---|
| RS037 | Un paquete de DATA, debe de incluir qué área lo envió, que tipo de dato es y los datos. |
| RS038 | Cuando un BN recibe un paquete DATA este debe ser redireccionado a algún SN. En caso de ser necesario se crearán más SN. |
| RS039 | Cuando se crea un SN, este inicia un nuevo proceso, el cual recibe como parámetros, su ID, el ID y dirección del BN. |
| RS040 | Al iniciar, el SN agrega a su tabla la dirección del BN y le manda un mensaje SINGINCHILD. |
| RS041 | Cuando un BN recibe un mensaje SINGINCHILD, éste lo agrega a su tabla de SN y en caso de ser necesario envía mensajes DATA pendientes. |
| RS042 | En caso de que un SN no cuente con la información de un área, almacenará el mensaje en una lista de espera. luego este enviara un mensaje SEARCHNODE, a su BN. |
| RS043 | Un BN al recibir un mensaje SEARCHNODE, verificará si tiene la información solicitada, en caso de tenerla contestara al nodo solicitante con un mensaje FINDNODE, en caso de no tenerla, almacenará la solicitud SEARCHNODE, en una lista de espera, luego enviará un mensaje SEARCHMULTICAST, para buscar el área solicitada. |
| RS044 | Cuando un BN recibe un mensaje SEARCHMULTICAST revisará si el es el área buscada, en caso de serlo enviará su información por medio de un mensaje FINDNODE. |
| RS045 | Cuando un BN recibe un mensaje FINDNODE, lo agrega a su tabla de ruteo y posteriormente envía los mensajes pendientes para este nodo. En caso de que este nodo haya sido solicitado previamente reenviara el mensaje FINDNODE. |
| RS046 | Cuando un SN recibe un mensaje FINDNODE, lo agregara a su tabla de ruteo y posteriormente envía los mensajes pendientes para este nodo. |

### 3.1.2. Diseño de software

Una vez se estabilizaron los requerimientos del equipo de investigación, estos se trasladaron al diseño de sistema. El diseño se documentó utilizando el estándar IEEE1016-2009. A continuación nombraremos los elementos principales que componen el sistema. Estos elementos y su relacion se pueden observar en la figura 3.1.

#### 3.1.2.1. Punto de vista de composición

Desde este punto de vista analizaremos las módulos que conforman al sistema.

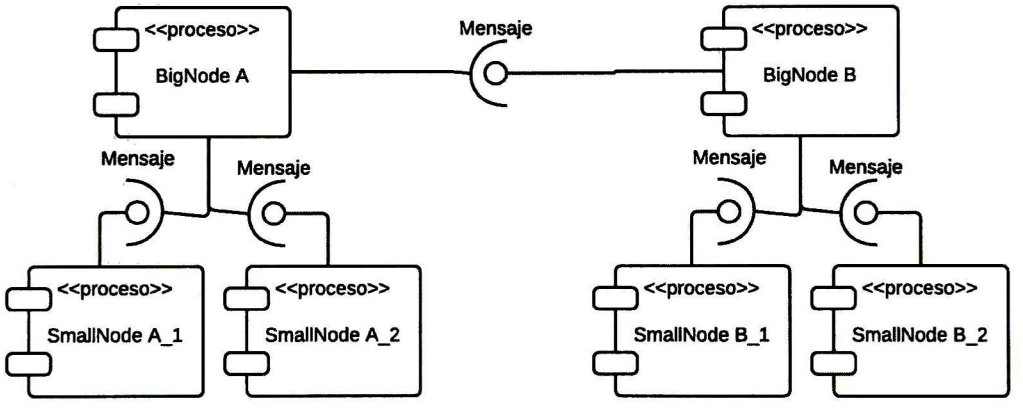



Figura 3.1: Arquitectura del middleware.

#### Big Node

Los BigNode's(BN) son los módulos principales del middleware, se encargan de la localización de otros BNs y el ruteo de todos los mensajes a través del sistema. Para esto se coordinan entre ellos para poder hacer una efectiva actualización de tablas de ruteo. Además administran a los SmallNode's(SN).

Un BN puede tener varios tipos de SNs, es decir pertenecer a una clase diferente.

### Small Node

Un SN es un módulo que se encarga de ejecutar un trabajo y reportar los resultados. Cada BN puede tener muchas instancias de un mismo SN y de esta manera poder paralelizar el trabajo.

### Estructura

Los BNs se pueden comunicar directamente entre ellos, y los SN, están bajo el control de un BN. Todos los nodos (BNs y SNs) se comunican a través de mensajes.

Un BN puede tener varios tipos de SN (cada tipo de SN es una clase que extiende a SmallNode), y a la vez puede manejar varias instancias del mismo tipo de SN, con el objetivo de repartir el trabajo entre ellos.

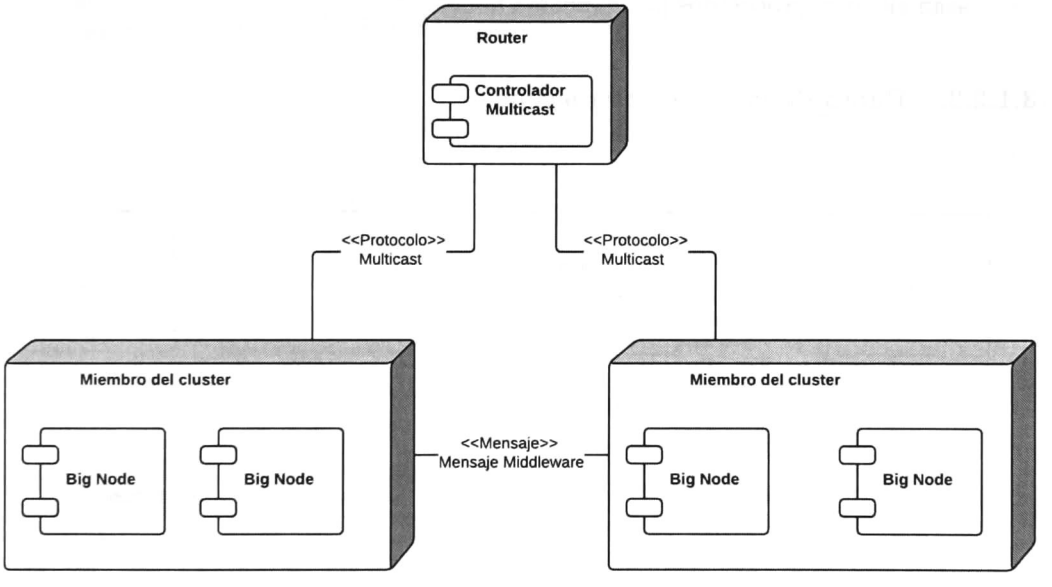



Figura 3.2: Diagrama de despliegue del middleware.

Mapeando esta estructura al plano físico ver (figura 3.2), un BN puede ubicarse en cualquier equipo dentro de la misma red, y cualquier equipo de la red puede tener un número indefinido de BN. Para intercambiar información entre ellos, los BN pueden utilizar multicast para enviar a todos los BN del sistema, además si se cuenta con la dirección de un miembro

específico, estos pueden comunicarse a través de una conexión p2p.

### Organización de SN

Ya se sabe que un BN puede tener varios tipos de SN, y que el número de cada tipo de SN puede variar para paralelizar el trabajo. Existen dos tipos de organización para los SN, dependiendo de las necesidades.

- Serial: Los nodos son organizados en una cola, y se les asigna trabajo conforme va llegando. Sí no existen suficientes nodos para realizar el trabajo requerido, entonces se crean mas.

  Mapped: Cada nodo esta asociado con una llave, y se tiene que usar esa llave para poder enviarle mensajes. Esta organización es útil si es necesario que ciertos mensajes sean siempre procesados por el mismo nodo.

### 3.1.2.2. Punto de vista de Interacción

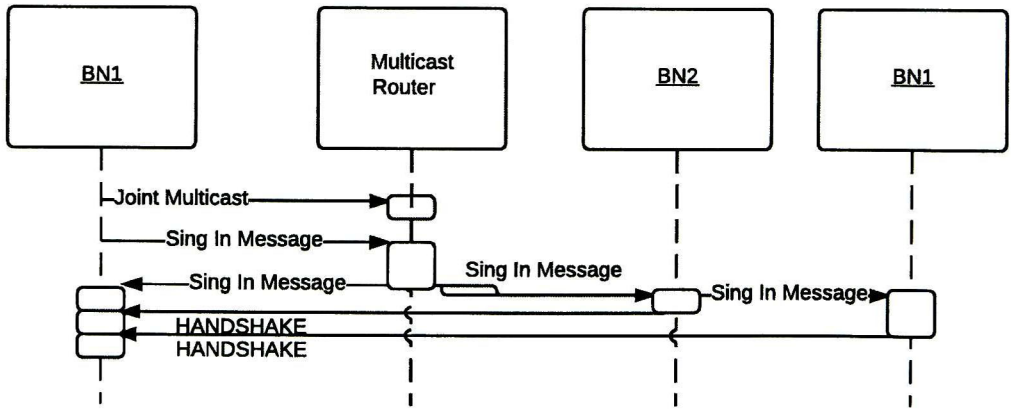



Figura 3.3: Diagrama de secuencia del protocolo SIA.

En esta sección analizaremos los diversos protocolos de interacción que son utilizados por los elementos del middleware y los propósitos que estos tienen.

### Protocolo Sing In Area

Cuando se inicia un BN, este se subscribe al grupo multicast e inmediatamente después, se presenta el mismo a los otros nodos con un mensaje *"Sing Iin Area"*(SIA), el cual contiene la dirección P2P del nodo. Los miembros del grupo multicast le contestan con un Handshake message, el cual contiene la dirección P2P del nodo.

### Protocolo Sing In Child

Cuando el BN recibe un mensaje de datos este puede ser enviado a un tipo de hijo. Al momento de enviarlo, puede ser posible que no existan hijos libres disponibles para poder procesar el mensaje. Por esto es necesario iniciar un nodo.

Cuando un SN es inicializado, este recibe como argumento la dirección IP y el puerto del nodo padre. Una vez que el padre recibe el mensaje *"Sing In Child"*(SIC), el cual contiene la dirección IP y puerto del hijo, el padre lo agrega a su lista de hijos libres, entonces procede a enviarle un mensaje de la cola de mensajes pendientes.

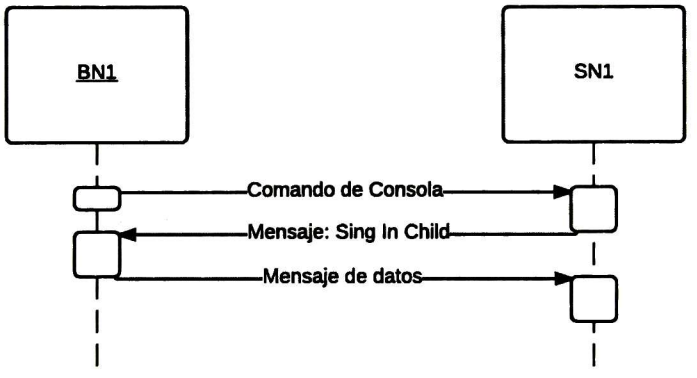



Figura 3.4: Diagrama de secuencia del protocolo SIC.

### Search Node Request

En caso de que un SN tenga que enviar un mensaje de datos a un BN y el SN no conozca la dirección del BN, el SN tiene que actualizar su información de ruteo, por lo tanto preguntará a su BN padre la dirección que el SN no conoce a través de un mensaje *"Search Node Request"*

En caso de conocerlo, como se observa en la figura 3.5 el BN le envía un mensaje *FindNode*, el cual contiene la dirección IP y el puerto del nodo que esta solicitando, es entonces que el SN procede a enviar la información.

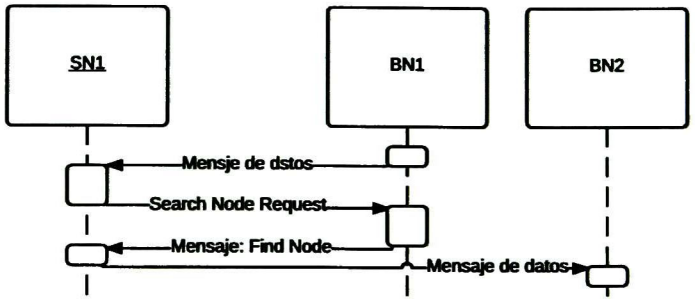



Figura 3.5: Diagrama de secuencia del protocolo SNR, el BN conoce la dirección que el SN solicita.

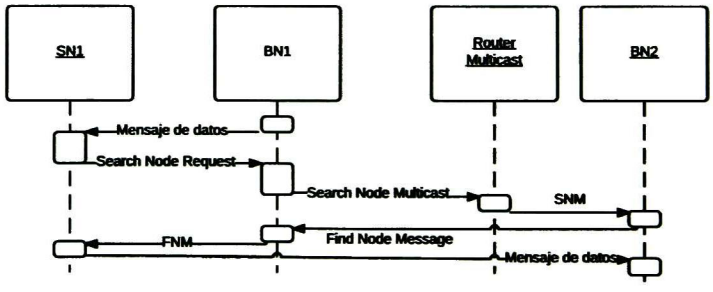



Figura 3.6: Diagrama de sequencia del protocolo SNR, el BN no conoce la dirección que el SN solicita.

Como se puede observar en la figura 3.6, en caso de que el BN no conozca la dirección que el SN solicita, este procede a solicitarla a través del protocolo multicast. Cuando el BN solicitado recibe el SNR, este envía su dirección al BN, este lo registrara y pasará la dirección al SN para que este envié el mensaje de datos.

Adicionalmente, también existe el caso de que el padre no conozca al nodo solicitado y que éste no exista en la red. Si esta situación llega a presentare, el nodo hijo reservará su mensaje en una cola, hasta que el nodo padre le envié la dirección y éste pueda continuar.

### Mensajes

Cada protocolo maneja un tipo específico de mensaje, llevando sólo la información necesaria para el funcionamiento del protocolo. Se ha requerido un estándar para la identificación de cada nodo del sistema, el identificador creado se denomina NODE ID. Un NODE ID contiene el Identificador de un BN, el tipo de SN y el numero de SN, con esto se puede identificar concretamente a cualquier nodo del sistema.

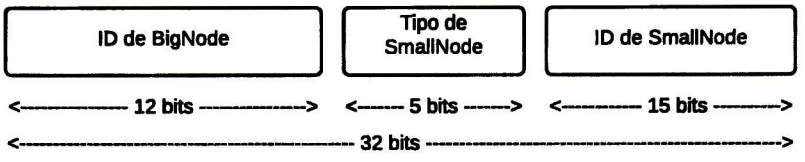



Figura 3.7: Estructura de NodeID.

### Mensaje SIA

El mensaje SIA (figura 3.8) esta compuesto por un código de operación, este le indica al middleware como desempaquetar el mensaje, posteriormente se encuentra el NODE ID, éste indica el Nombre del módulo que acaba de ingresar al sistema y después la dirección IP y el puerto P2P de este módulo.

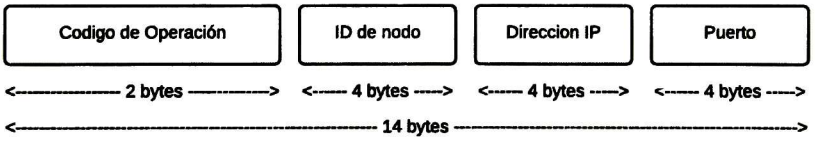



Figura 3.8: Estructura del mensaje SIA.

### Mensaje SIC

El mensaje SIC (figura 3.9) esta compuesto por un código de operación, este le indica al middleware como desempaquetar el mensaje, posteriormente se encuentra el Node ID, este

indica el Nombre del módulo que acaba de ingresar al sistema. La dirección IP y el puerto son obtenidos a través de la conexión por donde llego el mensaje.

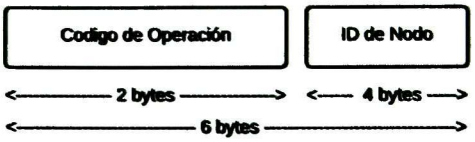



Figura 3.9: Estructura del mensaje SIC.

### Mensaje SNR

El mensaje SRN (figura 3.10) esta compuesto por un código de operación, este le indica al middleware como desempaquetar el mensaje, posteriormente se encuentra el NodeID, este indica el Nombre del modulo que acaba de ingresar al sistema. La dirección IP y el puerto son obtenidos a través de la conexión por donde llego el mensaje.

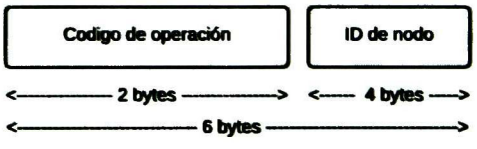



Figura 3.10: Estructura del mensaje SNR.

### Mensaje SNM

El mensaje SNM (figura 3.11) esta compuesto por un código de operación, este le indica al middleware como desempaquetar el mensaje, posteriormente se encuentra el NodeID, este indica el Nombre del módulo que acaba de ingresar al sistema. La dirección IP y el puerto son obtenidos a través de la conexión por donde llego el mensaje.

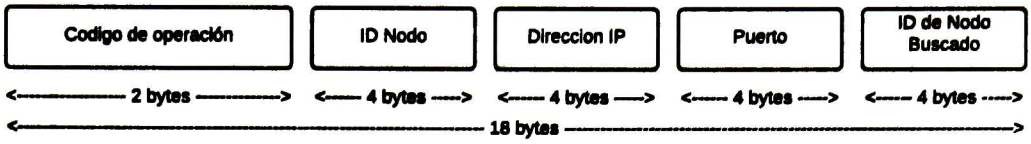



Figura 3.11: Estructura del mensaje SNM.

### Mensaje FN

El mensaje FN (figura 3.12) esta compuesto por un código de operación, éste le indica al middleware como desempaquetar el mensaje, posteriormente se encuentra el NodeID, este indica el Nombre del modulo que acaba de ingresar al sistema. La dirección IP y el puerto son obtenidos a través de la conexión por donde llego el mensaje.

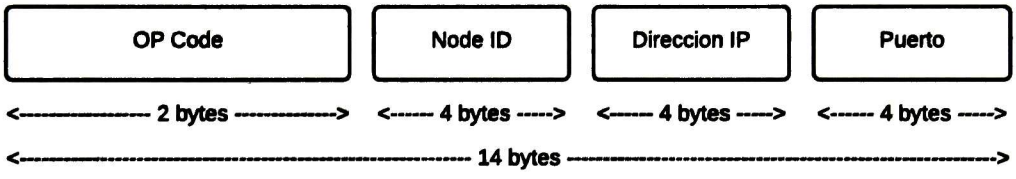



Figura 3.12: Estructura del mensaje FN.

#### 3.1.2.3. Punto de vista de estructura

### Carpetas

- LOG:

  En esta carpeta se almacenan las clases e interfaces necesarias para poder registrar de manera persistente la información obtenida de los nodos.

- Net:

  En esta carpeta encontramos toda la información concerniente a sockets y a comunicación de bajo nivel.

- P2P: Contiene las clases e interfaces necesarias para crear fácilmente una conexión P2P.
- Multicast: Almacena las clases e interfaces necesarias para poder registrarse y administrar las comunicaciones multicast de un nodo.
- Messages: Contiene clases con los distintos tipos de paquetes que el middleware maneja en sus distintos protocolos de comunicación.

Nodes: Aquí se encuentra toda la información de las entidades que conforman el middleware.

- BigNode: Es la entidad principal del middleware. Aquí se encuentran todos los protocolos que esta clase maneja.
  - NodeManager: Se encarga de administrar los SmallNode pertenecientes a un BigNode.
- SmallNode: Contiene las clases con el comportamiento básico de un SmallNode.

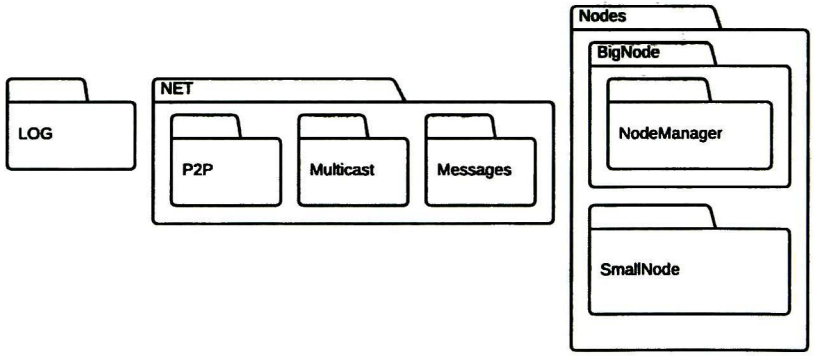



Figura 3.13: Diagrama de paquetes.

#### 3.1.2.4. Clases

### BigNode Class

Esta clase tiene que ser heredada a la implementación directa del usuario final, por tanto debe de enmascarar la comunicación y presentar métodos para realizar fácilmente las tareas fundamentales que el middleware requiere.

Metodos:

- `efferents(int nodeID, short dataType, byte[] data)`

  Este es el método que se encarga del envío de mensajes

  - nodeID: es el nombre del nodo al cual sera enviado el mensaje.
  - dataType: es un descriptor auxiliar de la información que puede ser utilizado por el usuario con el propósito de obtener fácilmente datos acerca del paquete.
  - data: es el paquete a ser enviado.

- `afferents(int nodeID, short dataType, byte[] data)`

  Este es el método donde se reciben los mensajes los parámetros consistirían en los mismos del método `efferents`.

  `addNodeType(int type, Class className)`:

  Se utiliza para que el BigNode agregue un nuevo tipo de SmallNode a administrar.

  - type: Especifica un identificador que sera utilizado para referirse al tipo de hijo en el futuro.
  - className: es la clase que este tipo de nodo hijo instanciara.

- `sendToCild(int type, int sender, short dataType, byte[] data)`:

  Se utiliza para enviar mensajes a un SmallNode de ese BigNode.

  - type: especifica el tipo de SmallNode al cual sera enviado.
  - Los siguientes 3 parámetros son idénticos al método efferents.

### BigNodeSocket Class

Esta clase se encarga de manejar los diferentes protocolos de comunicación de el BN. Cuenta con clases asociadas que se encargar de aspectos mas específicos, como conexiones multicast y P2P los cuales serán descritas posteriormente.

- `efferents(int nodeID, short dataType, byte[] data)`:

  Este es el método que se encarga del envío de mensajes, a este nivel consigue la dirección y lo envía por el puerto a trabes del método send.

  Sus parámetros son los que recibe de la clase BigNode

- `afferents(int nodeID, short dataType, byte[] data)`:

  Este es el método donde se reciben los mensajes. los parámetros consistirían en los mismos del método efferents.

  `send(Address addr, byte[] data)`: Se enviar de enviar la información a trabes del socket.

  - addr: Contiene la dirección IP y el puerto por el cual sera enviada la información.
  - data: Es la información que sera enviada en forma de cadena de bits.

  `receive(DatagramPacket datagramPacket)`: Recibe directamente el datagrama UDP del socket de escucha, su trabajo es desempaquetarlo y procesar la información hacia el BigNode.

  - datagramPacket: Es el paquete empaquetado con el formato del protocolo UDP.

- `startNode()`:

  Se encarga de levantar los puertos correspondientes para el bigNode y unirse al grupo multicast.

### SmallNodeManager Class

Esta clase se encarga de administrar las listas de direcciones de cada SN que esta registrado a este BN. Están separadas por tipo. Y cada lista puede tratar de manera diferente el balanceo de cargas.

Metodos:

- `addTye(int type, String className, NodeConfiguration options)`: Agrega el tipo de SmallNode y lo asocia con una clase, listo para ser instanciado.

  - type: Especifica un identificador que sera utilizado para referirse al tipo de hijo en el futuro.
  - className: es la clase que este tipo de nodo hijo instanciara.
  - options: Es un objeto que contiene todas las posibles configuraciones para un SmallNode.

### SmallNode Class

Esta clase SmallNode es la abstracción de la entidad homónima, esta presenta métodos para realizar las tareas que esta entidad necesita para su funcionamiento, tiene que ser heredada a la implementación directa del usuario final.

Metodos:

- `efferents(int nodeID, short dataType, byte[] data)`:

  Este es el método que se encarga del envío de mensajes, sus parámetros son idénticos al método `efferents` de la clase BigNode.

- `afferents(int nodeID, short dataType, byte[] data)`:

  Este es el método donde se reciben los mensajes los parámetros consistirían en los mismos del método `efferents`. En este método es donde se implementa el comportamiento del nodo.

### SmallNodeSocket

Esta clase se encarga de manejar los diferentes protocolos de comunicación de el SN. Cuenta con clases asociadas que se encargar de aspectos mas específicos, como conexiones P2P y manejo de un log, los cuales serán descritas posteriormente.

Métodos:

`efferents(int nodeID, short dataType, byte[] data)`:

Este es el método que se encarga del envío de mensajes, a este nivel consigue la dirección y lo envía por el puerto a través del método `send`.

Sus parámetros son los que recibe de la clase BigNodeSocket

- `afferents(int nodeID, short dataType, byte[] data)`:

  Este es el método donde se reciben los mensajes. los parámetros consistirían en los mismos del método `efferents`.

- `send(Address addr, byte[] data)`: Se enviar de enviar la información a trabes del socket.

  Sus parámetros son los que recibe de la clase BigNodeSocket

`receive(DatagramPacket datagramPacket)`: Recibe directamente el datagrama UDP del socket de escucha, su trabajo es desempaquetarlo y procesar la información hacia el BigNode.

Sus parámetros son los que recibe de la clase BigNodeSocket

- `startNode()`:

  Se encarga de levantar los puertos correspondientes para el SmallNode y registrase con su BigNode.

### P2P Class

La clase P2P proporciona un Hilo de escucha P2P independiente al hilo principal. Esta se encarga de recibir toda la comunicación directa a este puerto, sin que la clase que lo utilice se encargue en algún momento de gestionar directamente los sockets.

Métodos:

`send(Address add, byte[] m)`:

Se encarga de enviar la cadena de bytes $m$ a la dirección add.

`run()`:

Este metodo es donde se ejecuta el hilo, se encarga de escuchar los mensajes de entrada P2P. y enviarlo al objeto de escucha por medio del método receive de la interfase Nodeable

### Multicast Class

La clase Multicast proporciona un hilo de escucha multicast independiente al hilo principal. Esta se encarga de recibir toda la comunicación proveniente de este protocolo, sin que la clase que lo utilice se encargue en algún momento de gestionar directamente los sockets.

Métodos:

- `joinGroup()`:

  Se encarga de inicializar los sockets y unirse al grupo multicast.

- `leaveGroup()`:

  Se encarga de inicializar los sockets y unirse al grupo multicast.

- `send(byte[] m)`:

  Se encarga de enviar la cadena de bytes m a todos los miembros del grupo.

- `run()`:

  Este metodo es donde se ejecuta el hilo, se encarga de escuchar los mensajes de entrada por el protocolo multicast. y enviarlo al objeto de escucha por medio del método receive de la interfase MulticastAble

### Múltiples instancias

También existe la necesidad de que se pueda programar un módulo y este módulo pueda ser replicado para varias criaturas virtuales al mismo tiempo, cada una de estas criaturas es denominada entidad.

Como se observa en la figura 3.15, equipo puede alojar módulos de diferentes entidades y módulos de diferentes entidades pueden estar en diferentes equipo.

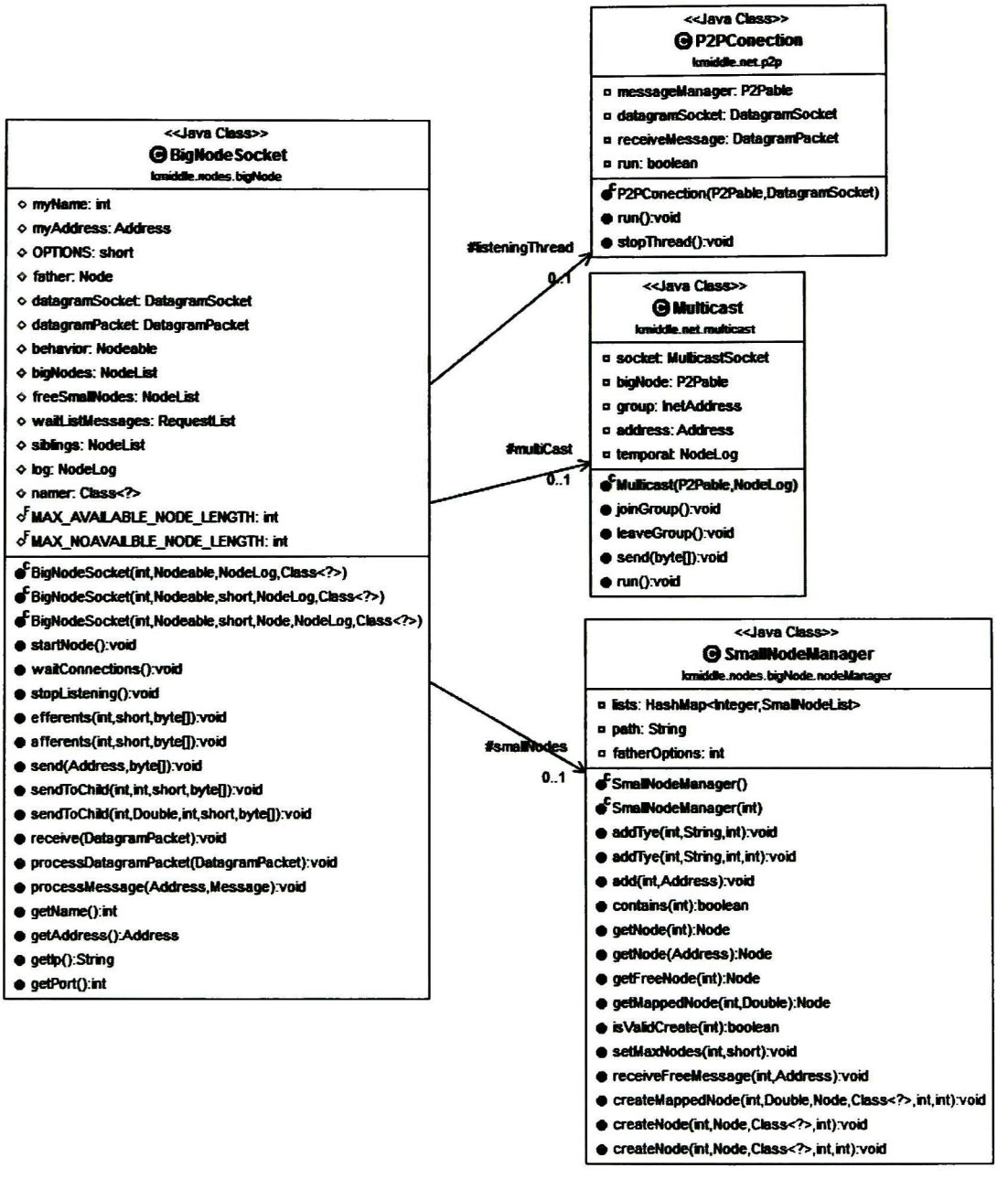



Figura 3.14: Clase BigNodeSocket.

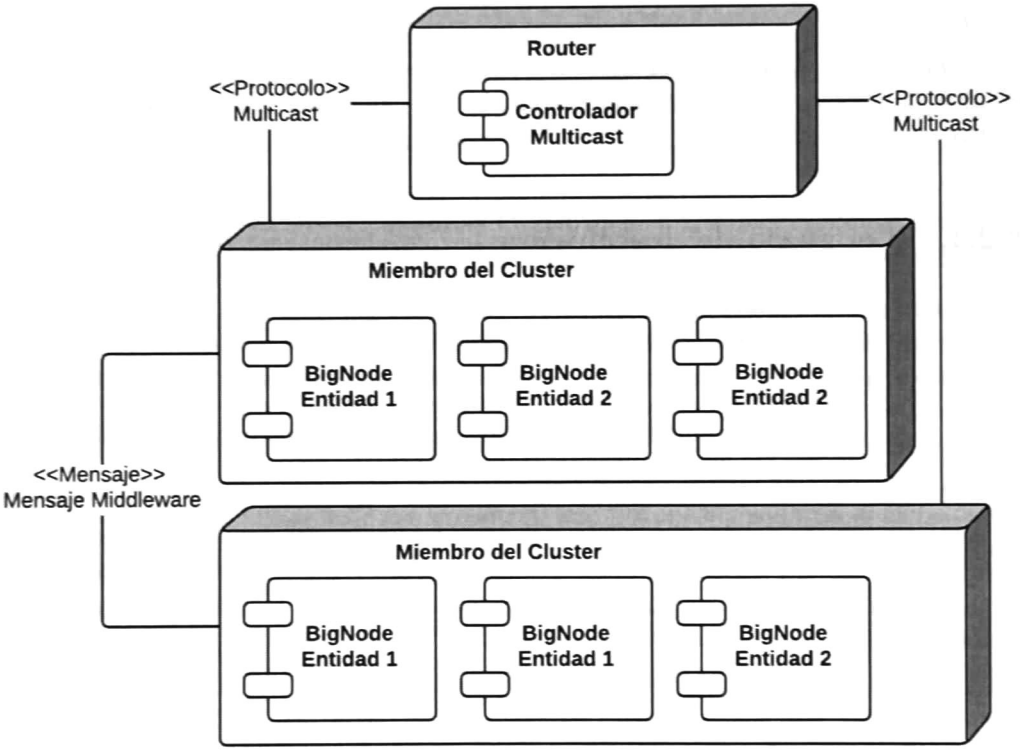



Figura 3.15: Dos entidades conviviendo en el mismo entorno.

## 3.2. Entorno de desarrollo integrado

Para la implementación de las herramientas se consideraron las opciones de crear un Entorno de desarrollo (*IDE por sus siglas en ingles*) desde cero, sin embargo esto seria muy poco conveniente por tiempo y calidad, En su lugar, se optó por adaptar uno preexistente, dado que actualmente existen plataformas robustas que proveen los elementos básicos que un IDE requiere y cuentan con una arquitectura abierta que permite modificaciones mediante plugins, complementos creados para añadir características o funcionalidades a software existente.

Se propone la adaptación de un IDE a través de la creación de un plugin especializado.El objetivo es que éste sea el encargado de generar el código fuente básico para la implementación de BN y sus respectivos SN.

### 3.2.1. Lenguaje de descripcion

Debido a que el plugin genera código fuente, se necesita poder administrar información de manera eficiente, por eso siempre es necesario contar con metadata, la cual son datos relacionado a cierta informacion, que no nesesariamente son parte de esta (ejemplo: lista de animales, metadata: fecha de creación de la lista), la cual nos puede recopilar las características que tendríamos que buscar en una gran cantidad de información.

Por este motivo desarrollamos un lenguaje de descripción de meta-data, basado en XML.

XML es un estándar muy flexible y concreto que permite almacenar información en un formato jerárquico. Ademas XML cuenta con muchas librerías y herramientas para su análisis y manipulación. Este lenguaje de descripción contiene la información mas relevante del modelo.

A continuación podemos observar un código de ejemplo, escrito en el lenguaje de descripción:

```
<Area name="Area1" X=10 Y=10>
  <Process name="1">
    <Input filter="data.type
                == tipo singular"/>
        <Output area="Area1"/>
  </Process>
  <Process name="2">
    <Input filter="data.type
                == patron de disparo"/>
        <Output area="Area2"/>
```

```
  </Process>
</Area>
<Area name="Area2" X=300 Y=30>
    <Process name="1">
          <Input filter="data.type
            == tipo singular"/>
          <Output area="Area1"/>
        </Process>
</Area>
```

Este lenguaje es un punto de abstracción intermedio entre el modelo y el código, el lenguaje provee información de cada BigNode, representado intermediamente como una área del cerebro. Las etiquetas mas importantes son:

- Area: Representa un area del cerebro y un BigNode en la implementación, este contiene toda la informacion asociada a la misma de una manera transparente para el usuario.
- Filter: El un filtro representa las reglas de ruteo, informa al BigNode que hacer con los paquetes de datos, a donde enviarlos entre los diferentes procesos.
- Process: Representa uno de los comportamientos de una area del cerebro en el modelo y dentro de la implementación representa a un SmallNode.
- In and Out: En el modelo representan las diferentes conexiones entre las área del cerebro, en la implementación, son limitantes que evitan que ciertas áreas del cerebro se comuniquen con otras, las cuales no están relacionadas.

Cuando el plugin inicia este debe de leer el archivo XML y transformarlo en un árbol XML el cual contiene la información y referencias a todos los archivos Java y las diferencies propiedades de configuración de cada componente del middleware.

El archivo XML debe de ser representado en el editor "lo que ves es lo que tienes"(*What You See What You Get (WYSWYG)*) y cambiar conforme al editor lo hace, creando editando y borrando áreas y procesos del modelo.

Cuando el plugin necesita mezclar implementaciones, cada modelo es representado con su árbol XML. Este árbol contiene todas la propiedades y referencias al código fuente, entonces los arboles son procesados y los posibles puntos de colisión son detectados, informados y reparados en caso de ser necesario.

### 3.2.2. Requerimientos

Los requerimientos del IDE fueron recopilados segun los problemas principales que se tenían con el middleware.

El IDE debe de ayudar con las funciones de organización de código.

- El IDE debe de contar con características para la corrección sintáctica y semántica de código.
- El IDE debe de presentar Herramientas para la depuración de código
- El IDE debe de automatizar el proceso de configuración del Middleware
- El IDE debe de proporcionar una interfaz WYSWYG para crear los modelos cognitivos.
- El IDE debe de mostrar las propiedades de los modelos cognitivos
- El IDE debe de proporcionas plantillas de código para los BNs y SNs.

Los requerimientos básicos (que no tienen que ver con el middleware) eran cubiertos por varias plataformas que existen en la actualidad. Entre ellas destacan por su amplia comunidad y apertura: **Eclipse** y **NetBeans**.

Nos decidimos por Eclipse debido a su gran uso en el área de desarrollo de software ademas de su gran comunidad de contribuidores y su apertura para el desarrollo de Plugins.

### 3.2.3. Documento de diseño de software

Para realizar el plugin se utilizo la API de Eclipse PDE, la cual proporciona métodos para poder hacer una integración sencilla a la plataforma Eclipse y otras librerias.

#### vista de diseño de estructura

#### Eclipse PDE

Eclipse es una plataforma que lleva muchos años en desarrollo y que no ha dejado de crecer, siempre incorporando nuevos elementos a su repertorio. Un vistazo simple de su estructura se puede reflejar en módulos (figura 3.16).

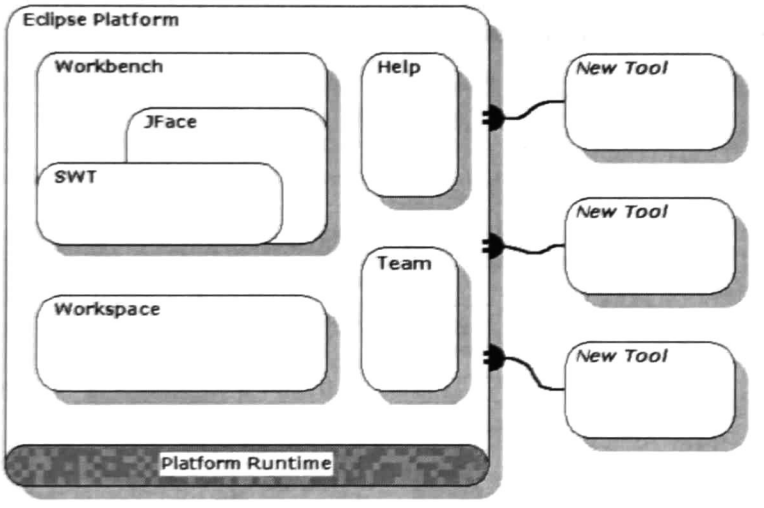



Figura 3.16: Arquitectura simplificada de eclipse [34].

- **Workspace**: se encarga de almacenar la información de configuración para el espacio de trabajo.
- **Workbench**: Da herramientas para la creación de la interfase gráfica desarrollando su propio Framework JFace, el cual esta escrito con SWT para que puedan trabajar en conjunto sin inconvenientes de compatibilidad.
- Las implementaciones de plugins, se hacen de forma sencilla debido a que estos son componentes externos a la plataforma que utilizan las funciones de la misma como

un servicio, de esta manera ambos pueden funcionar sin tener que hacer un fuerte acoplamiento entre ambos.

### Plugin Architecture

- Event Listener/sender: Este componente es el que se encuentra en contacto con los servicios que Eclipse proporciona para la integración. Recibe y envía los eventos que son ejecutados por el Plugin.
- Graph: Es la representación gráfica de las áreas del cerebro. Provee herramientas para crear, editar y eliminar BigNodes y SmallNodes.
- Validator: Una vez que el grafo recibe la indicación de hacer una modificación, este modulo revisa la meta-data de los archivos del proyecto y valida que los cambios pueden realizare, entonces envía la información a Code Generator.
- Code Generator: Con la información ya Validada, este procede a modificar los archivos de código Java, para ajustarse a la configuración del grafo.

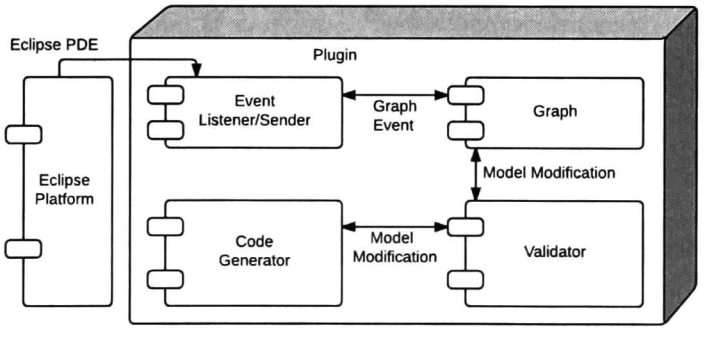



Figura 3.17: Diagrama de componentes de la arquitectura del plugin

# Capítulo 4

# Resultados

En este capítulo describimos los productos obtenidos de la implementación de un sistema que satisface las especificaciones presentadas en el capitulo anterior. Incluye un análisis de desempeño del middleware, a través de varias pruebas. Además se presenta un código de ejemplo para ilustrar la estructura básica de las clases principales del middleware.También incluye capturas de pantalla asi como una descripción de los componentes de la interfase gráfica del IDE para facilitar tanto su entendimiento como su uso.

## 4.1. Resultados del Middleware

Después de la fase de implementación y pruebas del middleware, se obtuvó un producto estable que satisface los requerimientos estipulados en el capítulo 3. El middleware se utiliza como una librería que provee clases para iniciar módulos de un sistema distribuido. Cada área del cerebro dentro del modelo cognitivo es representada por un BigNode, y los procesos cognitivos dentro de estas áreas son representados por SmallNodes.

Antes de poder implementar un BigNode, se requiere contar con una lista de ID no ambigua, esto se hace por medio de un archivo de configuración. Este es una clase que incluye constantes con los nombres de la áreas y sus respectivos ID's, para nuestro ejemplo usaremos la siguiente configuración.

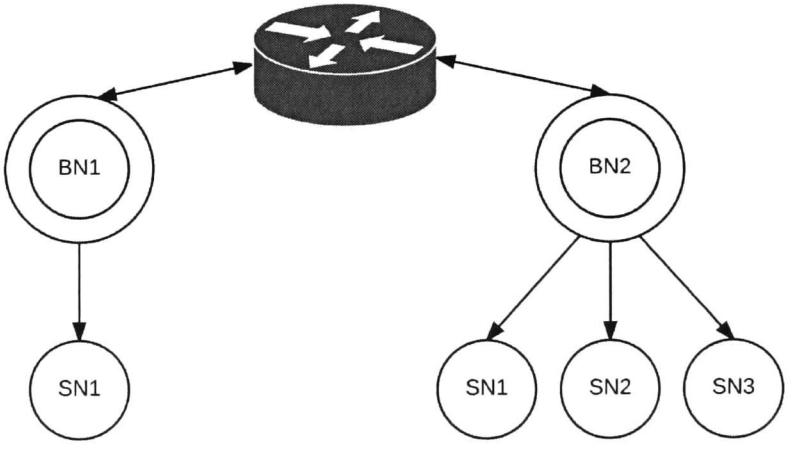



Figura 4.1: Estructura del caso de ejemplo.

Para este caso nuestro archivo de configuración seria:

```
package config;

public class AreaNames {

        public static int BigNode1       = 1048576;
        public static int BigNode1_proceso1     = 1;
        public static int BigNode2       = 2097152;
        public static int BigNode2_proceso1     = 1;
        public static int BigNode2_proceso2     = 2;
        public static int BigNode2_proceso3     = 3;

}
```

Los números que corresponden a los BigNodes, pueden parecer aleatorios, pero si recordamos la estructura de un NodeID (Software Description Document), el ID de un BigNode, son los primeros 12 bits de una cadena de 32, por lo tanto cada número correspondiente al AreaNames es '$n$ recorrido 20 bits a la izquierda (n << 20).

También puede parecer que los IDs de los procesos se repitan, sin embargo, la numeración de sus procesos es independiente para cada BigNode.

Una vez que se tiene el archivo de configuración, para implementar un BigNode basta con heredar de la clase BigNode y sobrescribir los métodos requeridos.

```
import kmiddle.nodes.bigNode.BigNode;

public class BigNode1 extends BigNode{

        public BigNode1(int name, NodeConfiguration config){
                super(name, config    AreaNames.class);
        }

        public void init(){
                addNodeType(AreaNames.BigNode1_proceso1
                        BigNode1_proceso1.class);
        }

        @Override
        public void afferents(int senderID  short dataType,
                byte[] data) {
        };
}
```

Con el código listado en la parte de arriba, ya se cuenta con un módulo funcional del sistema, disponible para enviar y recibir mensajes de con los otros módulos. Esto soluciona el problema de la infraestructura. El comportamiento del módulo esta listo para ser implementado por el investigador.

La funcion afferents se encarga de recibir todos los mensajes de datos que llegan a este BigNode, mientras que la funcion addNodeType indica al Nodo cuales son los tipos de SmallNodes que tiene disponibles y que ID le corresponde a cada uno.

Ahora bien, para implementar un SmallNode, el código es similar, solo que ahora, hereda de la clase SmallNode, la lógica del procesamiento se implementa en la función **Afferents**, la cual recibe los mensajes de entrada. El código del SmallNode sera ejecutado por múltiples instancias, dependiendo de cuantas solicitudes se reciben.

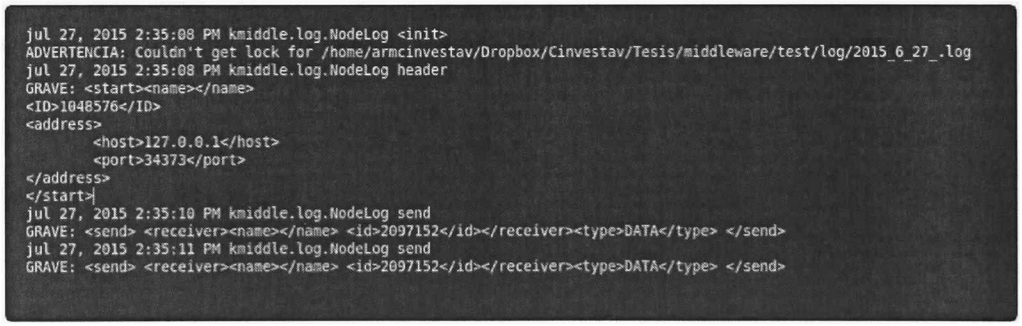



Figura 4.2: Captura de pantalla de la interface del middleware.

### 4.1.1. Confianza de la conexion y Thougtput

Para poder determinar la confianza y el Throughput máximo del sistema se tuvieron que realizar pruebas de rendimiento.

Las características de los equipos para las pruebas son los siguientes:

- Procesador: Intel i7-3770k
- HD: 300 GB
- RAM: 3904 MB

- Grafica: Nvidia GT 520
- Red: Realtek PCI GigaBit Ethernet

La red en la que se realizaron las pruebas pudo soportar un througtput máximo de **1287 MB** mediante envió de datos en una conexión serial peer-to-peer.

#### 4.1.1.1. Serie de pruebas 1

Esta serie de pruebas fue realizada con los siguientes parametros:

- Numero de nodos: 2

  Numero de nodos por equipo: 1
- Tamaño de cada paquete: 4 B
- Limite de la conexión: 1 GB/s

| Mensajes/segundo | KB enviados | Througput | % de recepción de los mensajes |
|---|---|---|---|
| 860 | 3KB | 3KB | 1 |
| 1730 | 7KB | 7KB | 0.99997 |
| 3410 | 13KB | 13KB | 0.99994 |
| 6578 | 26KB | 26KB | 0.99992 |
| 7713 | 30KB | 30KB | 0.9996 |
| 10014 | 39KB | 39KB | 0.99772 |
| 24055 | 100KB | 94KB | 0.9421 |
| 334713 | 21618KB | 1307KB | 0.060481 |

Tabla 4.1: Prueba de desempeño 1.

La confianza en la transmisión se mantiene muy estable (por encima del 99.5 %), alcanzando una conmutación de 10,014 mensajes por segundo entre 2 nodos, sin embargo como cada paquete pesa 4 Bytes, el Througput alcanzado es solo de **39KB/s**, lo cual es muy bajo considerando que el limite de la conexión es 1 GB/s.

Sin embargo al aumentar el numero de mensajes por segundo el precio pagado por la conmutación de muchos paquetes pequeños, hace que el ruteador se sature y empiece a perder paquetes.

#### 4.1.1.2. Serie de pruebas 2

Esta serie de pruebas fue realizada con los siguientes parámetros:

Número de nodos: 2

- Número de nodos por equipo: 1
- Tamaño de cada paquete: 60 KB

| Mensajes/segundo | KB enviados | Througput | % de recepción de los mensajes |
|---|---|---|---|
| 1730 | 101368KB | 101367KB | 0.999989919 |
| 3410 | 200139KB | 199805KB | 0.998330081 |
| 6578 | 399058KB | 385430KB | 0.96585 |
| 7713 | 529805KB | 451934KB | 0.853018182 |
| 10014 | 1185834KB | 586758KB | 0.494805857 |

Tabla 4.2: Prueba de desempeño 2.

La confianza en la transmisión se mantiene muy estable (por encima del 99.5 %), alcanzando una conmutación de 3410 mensajes por segundo entre 2 nodos. Para esta serie de pruebas se usaron paquetes de 60KB (el limite teórico de un paquete UDP, es 64KB), obteniendo un Througput de 199.8 MB/s y una confianza superior al 99.5

Se pudo conseguir enviar 6578 mensajes/s, con un Througput de 385.43 MB/s sin embargo la confianza decayó hasta 96.5

Se intento saturar el canal para ver cual era el Througput máximo y la confianza que este tenía, se lograron enviar 10,014 mensajes/segundo consiguiendo un Througput de 1,185 MB/s sin embargo la confianza es de 49.4 %, representando una perdida de más de la mitad de los paquetes.

#### 4.1.1.3. Serie de pruebas 3

Número de nodos: 4

- Número de nodos por equipo: 2
- Tamaño de cada paquete: 60 KB

Para esta prueba se utilizaron 4 nodos, 2 en cada equipo y solo se comunicaron en parejas (Nodo A con Nodo B y Nodo C con Nodo D).

| Mensajes/segundo | KB enviados | Througput | % de recepción de los mensajes |
|---|---|---|---|
| 1720 | 100784KB | 100781KB | 0.9999697568 |
| 3460 | 202746KB | 202738KB | 0.9999596766 |
| 6820 | 400296KB | 399623KB | 0.998317506 |
| 13156 | 797783KB | 770898KB | 0.9663 |
| 20028 | 2378427KB | 1173594KB | 0.4934328358 |

Tabla 4.3: Prueba de desempeño 3.

El Througput se obtuvo de la suma de la transferencia de ambas parejas y la confianza se obtuvo del promedio de la confianza de ambas.

La confianza en la transmisión se mantiene muy estable (por encima del 99.5 %), alcanzando una conmutación de 6820 mensajes/segundo en el sistema. Para esta serie de pruebas se usaron paquetes de 60KB, obteniendo un Througput de 399.62 MB/s y una confianza de 99.8

## 4.2. Resultados Plugin

Después de haber seguido los pasos estipulados en el SDD del plugin se consiguió crear la herramienta, la cual administra los templates de códigos y la configuración del middleware.

Los procesos de instalación y manual de usuario están incluidos en el anexo.

### 4.2.1. Características

#### Representación de modelos cognitivos

A través de una interfaz WYSWYG se recrean los modelos cognitivos, las áreas involucradas son nombradas y se añaden los diferentes procesos que las conforman, todo esto a trabes de una sencilla interface.

#### Generación automática de código

Conforme se va modificando el diagrama cognitivo, las clases van apareciendo en el explorador de paquetes, ya configuradas y listas para que el usuario empiece a implementar el código.

#### Manejo de la configuración

El plugin cuenta con una ventana de propiedades la cual muestra los diferentes atributos y valores de los elementos del middleware, estos pueden ser configurados de manera sencilla por medio de esta ventana.

### 4.2.2. Ventanas del plugin

1. Esta es el área de editor WYSWYG y sus herramientas, aquí se permite crear y eliminar áreas, procesos y conexiones.
2. Esta es la representación de un área del cerebro, en este punto su codigo fuente y configuración ya han sido generados.
3. Aqui se muestran todos los proceso que maneja esta área del cerebro cada proceso tiene su propio código fuente y es implementado por el usuario.
4. Las conexiones marcar el flujo valido de información, no se puede enviar información entre 2 áreas si no existe un camino valido anteriormente.

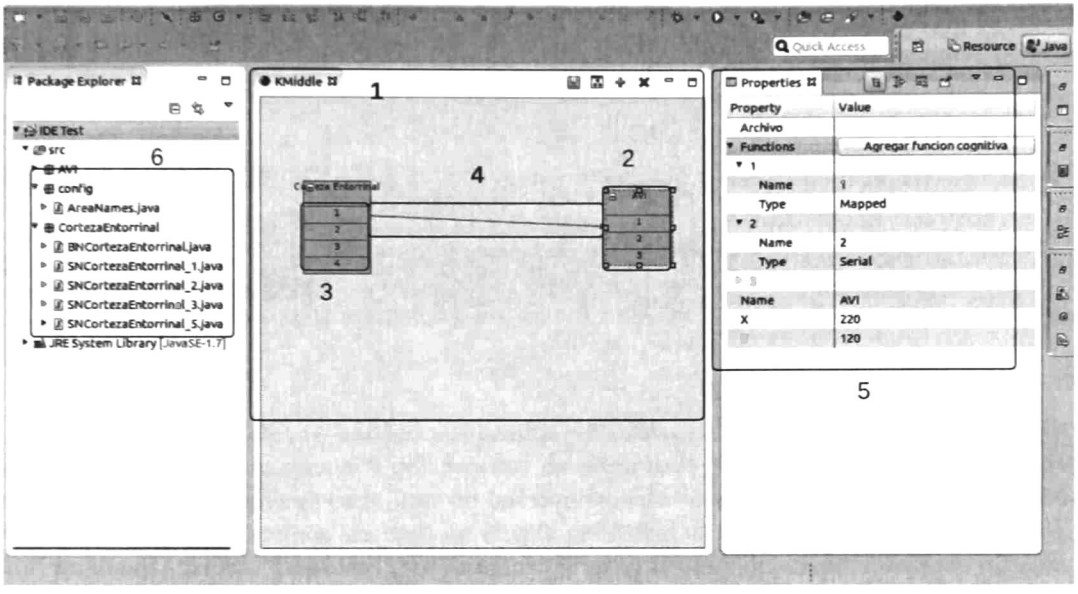



Figura 4.3: Captura de pantalla del Plugin.

5. Uno de los componentes mas importantes del plugin: la ventana de propiedades. aquí se puede modificar cada aspecto del código fuente de una manera fácil y organizada.

6. Los archivos de código fuente que el Plugin a generado automáticamente, pueden ser editados directamente por el usuario.

# Capítulo 5

# Casos de uso

El middleware y el IDE, ya han sido utilizados exitosamente por el grupo de investigación a través de las varias iteraciones del proceso de desarrollo de software, en este capítulo detallaremos como se realizó cada una de las implementaciones y los resultados obtenidos de las mismas. Presentaremos un caso en el que se utiliza el middleware junto con el IDE en un ambiente virtual. También abordaremos un caso de uso del middleware sin el IDE. Y un ejemplo de la utilización del middleware en un sistema multiagentes sin relación con arquitecturas cognitivas.

## 5.1. Implementación con un ambiente virtual

Esta implementación es detallada en el articulo "A Motivational Model of Hunger for a Cognitive Architecture"[1]. En el articulo se presenta un modelo bio-inspirado para generar, mantener y eliminar necesidades relacionadas con el hambre.

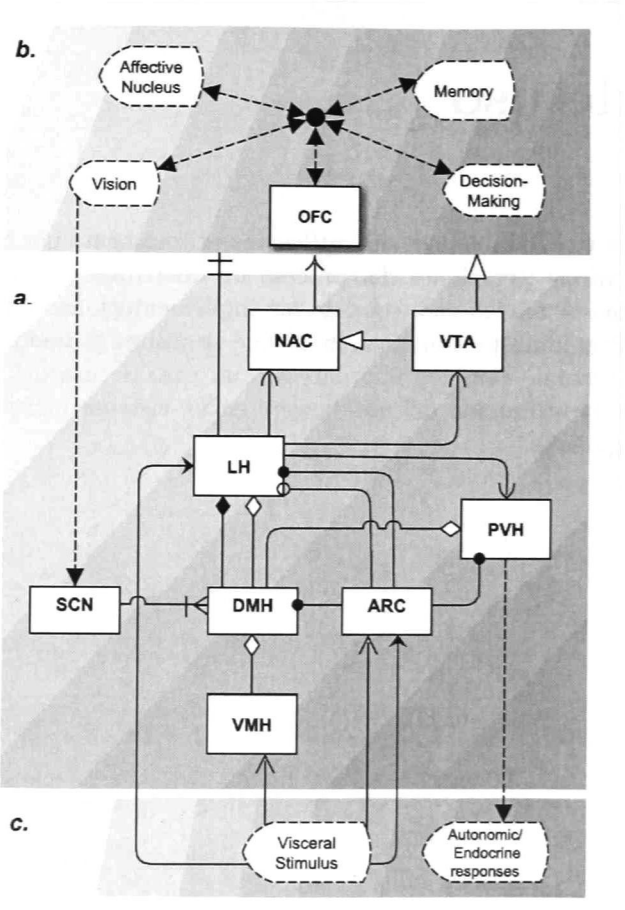



Figura 5.1: Captura de pantalla de la implementación en el plugin - Imagen con permiso de [1]

Como se puede ver en la figura 5.1, el modelo cognitivo cuenta con varias áreas o módulos. Este modelo contiene un flujo de información y un comportamiento establecido. El modelo es recreado en el IDE y adicionalmente son agregados los procesos que están involucrados en

cada área.

Una vez que el diagrama es trasladado al IDE, todas clases y la configuración se generan automáticamente. Por lo que la tarea en la que se avocan los investigadores es programar el comportamiento de cada área y proceso del sistema según lo establecen sus modelos.

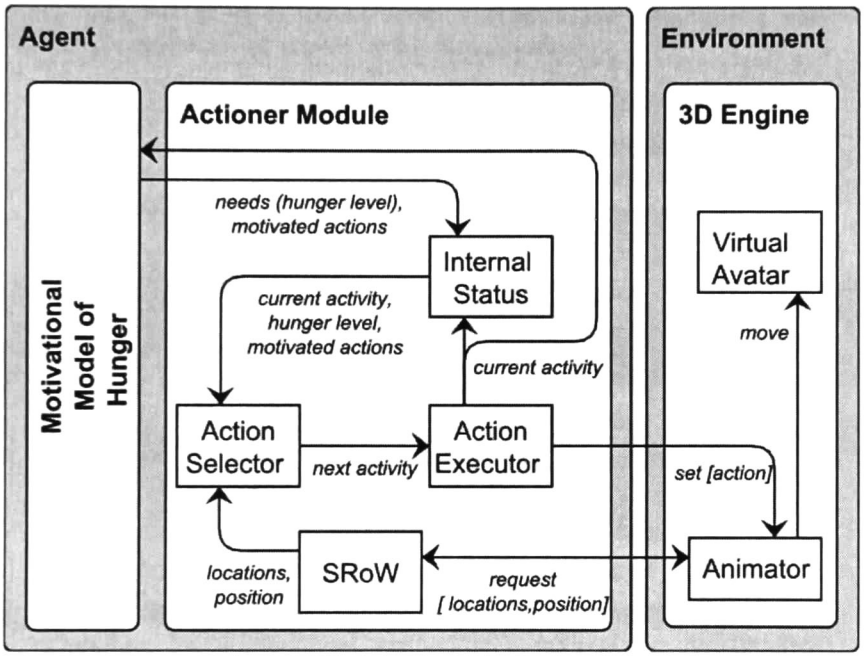



Figura 5.2: Diagrama completo de la implementación, incluyendo módulos externos - Imagen con permiso de [1]

Como se puede observar en la figura 5.2, en esta implementación se utilizó una conexión con un entorno virtual, sin embargo la comunicación nativa del sistema con módulos externos (i.e el ambiente virtual) aun esta en desarrollo, por lo que los investigadores tuvieron que usar una conexión serial pre-configurada, para poder comunicarse con el motor gráfico.

Para esta implementación se utilizaron paquetes de datos menores a 10KB y se enviaron paquetes cada 2 segundos por lo que no existe la posibilidad de que los canales se saturen de datos con una cantidad tan pequeña de información.

Aunque la conexión serial envía paquetes de tamaño considerable, esta conexión se hace de manera independiente a los protocolos del middleware, por lo que no se incluye en las mediciones de desempeño.

Los resultados de los experimentos fueron consistentes con las suposiciones y el middleware desempeño su función sin inconvenientes.

## 5.2. Implementación usando solo el middleware

Esta implementación es detallada en el articulo "Middleware for Integrating Cognitive Architectures"[35]. En este artículo se hace una descripción general del middleware y sus ventajas, ademas proporciona un ejemplo de implementación, basado en un experimento de neurociencias.

En la figura 5.3 se observa el modelo cognitivo de habituación a la novedad, con sus respectivas áreas, cada una de estas tiene que ser abstraida a un BigNode y posteriormente sus procesos son implementados al través de un SmallNode.

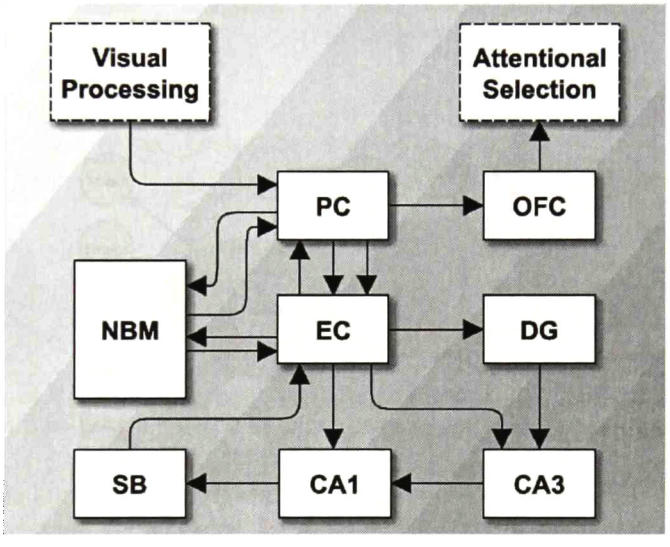



Figura 5.3: Modelo cognitivo de habituacion a la novedad

Como esta implementación fue realizada sin utilizar el IDE, el proceso de abstracción y configuración se realizó manualmente. El investigador configuro las áreas y los procesos de la manera que se ejemplifica en la figura 5.4. Cada área del modelo cognitivo fue asociada a un BigNode y los diferentes procesos que maneja cada área fueron implementados por medio de uno o mas SmallNodes.

Al igual que el caso anterior el middleware desempeño su función sin inconvenientes y los resultados de los experimentos fueron coherentes con las suposiciones.

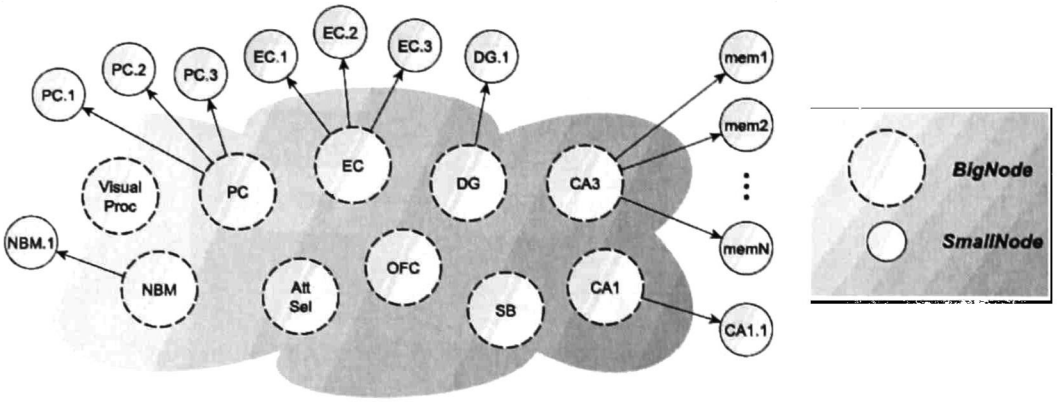



Figura 5.4: Adaptación del modelo cognitivo a una implementación modular del middleware

## 5.3. Implementacion en Sistema multi agentes

Aunque el middleware tiene un enfoque en arquitecturas cognitivas, también se puede usar como un middleware de proposito general en el area de sistemas multiagentes, ya que cuenta con todos los elementos para proporcionar una infraestructura de comunicación eficaz.

La implementación se realizo utilizando solamente el middleware, cada BigNode represento un agente del sistema y los SmallNodes, los diferentes procesos que cada agente realiza.

En este Sistema multiagente tiene como función el proveer una serie de servicios para solucionar las necesidades del usuario a través de internet. Cada agente tiene tareas específicas como procesar instrucciones y aprender de ellas, buscar nuevos servicios u opciones para tareas ya existentes.

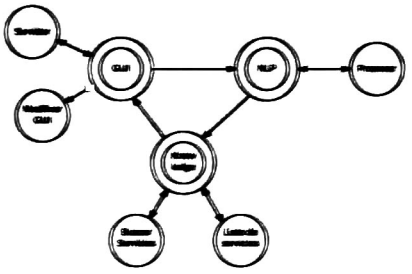


Figura 5.5: Estructura de la implementacion del sistema multiagentes

Como se puede apreciar en la figura 5.5 l usuario utiliza un navegador web como interfaz gráfica para el consumo de servicios. El sistema cuenta con un servidor web ligero para la comunicación con el navegador, el agente GUI se encarga de obtener las entradas y proveer las salidas. asi como de tambiÉn mejorar constantemente la interface de usuario, conforme esta se usa. GUI envía las entradas a NLP, el cual se encarga de asociar acciones a procesos y mandarlos ejecutar en Knowledge. Knowledge contiene la información del sistema así como los procesos, este recibe los datos ya formateados por parte de NLP.

La intensidad del flujo de datos es determinada por las solicitudes del usuario. sin embargo estas se limitan a instrucciones de pocos KB. siendo muy improbable una saturacion del canal.

# Capítulo 6

# Conclusiones

Después de haber seguido un adecuado proceso de desarrollo de software, utilizando los estándares de la industria, se consiguió un producto estable, que satisface los requerimientos establecidos en la etapa de diseño.

Podemos afirmar lo último debido a que la implantación se probó en múltiples implementaciones. Además el entorno de desarrollo propuesto es útil durante el proceso que requiere pasar el diagrama cognitivo a una implementación y configurar los diferentes módulos. Por último, también podemos afirmar que el middleware es útil en la implementación de sistemas multi-agentes como fue mostrado en el caso de uso del capítulo 5.

Finalmente, durante nuestra investigación en el área de neurociencias nos hemos dado cuenta de que existe un sistema el cual realiza tareas que son efectuadas por nuestro sistema. Este componente del cerebro es denominado "sistema limbico". Este consiste de una intersección de varios sistemas y a la ves es el punto de integración entre diferentes funciones cognitivas del sistema. Es nuestro objetivo en el futuro cercano realizar una extensa investigación de este sistema para decidir si este sistema cumple con los requerimientos en los que se basó nuestra propuesta presentada en el presente documento. Y si esto es cierto entonces estudiar como las bases neurocientificas satisfacen cada uno de los requerimientos presentados en esta disertación y aun más que otros requerimientos satisface este sistema limbico y como son efectuadas las tareas necesarias para que estos sean satisfechos.

## Trabajo futuro

El estado del middleware actual es estable, aunque aun hay características que son necesarias para satisfacer ciertos escenarios. Algunas de estas caracteristicas son enumeradas a continuación:

- Comunicación nativa con sistema ambiente: Lo que se intenta evitar es hacer una im-

plementación para un motor gráfico especifico y de esta manera restarle flexibilidad al middleware. Para resolver este problema es necesario crear un protocolo para la comunicación con módulos externos, ya sea un motor gráfico u otro software. De esta manera será posible intercambiar información entre el middleware y procesos externos (diferentes motores gráficos).

- Superar el limite teórico de los paquetes UDP: En los requerimientos se estipularon paquetes con tamaños reducidos (inferiores a 60k). No obstante en las pruebas se ha requerido enviar paquetes de mayor tamaño (imágenes), por lo que se debe de crear un protocolo de segmentación e integración de mensajes para poder enviar más información y que sea recibida como un solo mensaje por las capas superiores del middleware.

- Integración automática: El IDE puede configurar proyectos y generar código automáticamente, sin embargo la tarea de integración entre diferentes implementaciones aun se realiza manualmente. Se desea que el plugin administre esto automáticamente a través de la red y sincronice la configuración.

- Utilización de librerías gráficas nativas: En la implementación del IDE se utilizaron librerias externas para el apartado de grafos. Sin embargo esto requirió que los componentes fueran embebidos, causando un bajo desempeño en los tiempos de actualización. Se pretende corregir esto con la utilización de la librería nativa: ZEND.

Estas son solo algunas de las líneas que consideramos puede tomar el trabajo aquí presentado para mejorar su desmpeño tanto en tiempo como en facilidad de utilización. Sin embargo es posible que durante su uso esta lista se incremente. Sin embargo debido a la arquitectura del sistema, debiera de ser relativamente fácil realizar estas mejoras o cambios.

# Bibliografía


[1] C. A.-C. Myrna S. Zamarripa, Daniel Madrigal and F. Ramos, "A motivational model of hunger for a cognitive architecture," 2015.

[2] D. E. Kieras and D. E. Meyer, "An overview of the epic architecture for cognition and performance with application to human-computer interaction," *Hum.-Comput. Interact.*, vol. 12, pp. 391–438, 1997.

[3] J. E. Laird, "Extending the soar cognitive architecture," *Frontiers in Artificial Intelligence and Applications*, vol. 171, p. 224, 2008.

[4] P. Langley, J. E. Laird, and S. Rogers, "Cognitive architectures: Research issues and challenges," *Cognitive Systems Research*, vol. 10, no. 2, pp. 141–160, 2009.

[5] OMG, "The common object request broker: Architecture and specification," *OMG Document Number 91.12.1*, 1991.

[6] S. Franklin and F. G. P. Jr., "The lida architecture adding new models of learning to an intelligent, autonomous software agent," *Integrated Design and Process Technology*, vol. 703, pp. 764–1004, 2006.

[7] G. Metta, P. Fitzpatrick, and L. Natale, "Yarp: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, pp. 43–48, 2006.

[8] G. A. T. Blanco, *Metodologia para el desarrollo de arquitecturas cognitivas basada en neurociencias.* 2013.

[9] J. E. Laird, *The Soar Cognitive Architecture.* The MIT Press, 2012.

[10] S. Profanter, "Cognitive architectures," in *HaupteSeminar Human Robot Interaction.[II] Marion G. Ceruti, Senior Member IEEE (2000)."The Relationship Between Artificial Intelligence and Data Mining: Application to Future Military Information Systems". Published in: Systems, Man and Cybernetics, 2000 IEEE international Conference*, vol. 3, 2012.

[11] W. Duch, R. J. Oentaryo, and M. Pasquier, "Cognitive architectures: Where do we go from here?," in *AGI*, vol. 171, pp. 122–136, 2008.

[12] J. F. Sowa, "Conceptual structures: information processing in mind and machine," 1983.

[13] D. E. Rumelhart, J. L. McClelland, P. R. Group, *et al.*, "Parallel distributed processing: Explorations in the microstructures of cognition. volume 1: Foundations," *MIT Press, Cambridge, MA*, vol. 2, pp. 560–567, 1986.

[14] R. Sun and F Alexandre, *Connectionist-symbolic integration: From unified to hybrid approaches.* Psychology Press, 2013.

[15] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems. Concepts and Designs*, ch. Characterization of Distributed Systems, pp. 1–36. Addison Wesley, 5 ed., 2012.

[16] P. A. Bernstein, "Middleware: a model for distributed system services," *Communications of the ACM*, vol. 39, no. 2, pp. 86–98, 1996.

[17] D. Bakken, "Middleware," 2001.

[18] A. S. Tanenbaum and M. Van Steen, *Distributed systems.* Prentice-Hall, 2007.

[19] A. Puder, K. Romer, and F. Pilhofer, *Distributed Systems Architecture A Middleware Approach*, ch. Introduction, pp. 1–6. Elsevier Morgan Kaufmann Publishers, 2006.

[20] J. Siegel, "Omg overview: Corba and the oma in enterprise computing," *Communications of the ACM*, vol. 41, no. 10, pp. 37–43, 1998.

[21] N. Hawes, M. Zillich, and J. Wyatt, "Balt & cast: Middleware for cognitive robotics," in *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on*, pp. 998–1003, Aug 2005.

[22] N. Hawes, J. Wyatt, and A. Sloman, "An architecture schema for embodied cognitive systems," *SCHOOL OF COMPUTER SCIENCE RESEARCH REPORTS-UNIVERSITY OF BIRMINGHAM CSR*, vol. 12, 2006.

[23] E. Curry, "Message-oriented middleware," *Middleware for communications*, pp. 1–28, 2004.

[24] G. Sandini, G. Metta, and D. Vernon in *50 Years of Artificial Intelligence* (M. Lungarella, F. Iida, J. Bongard, and R. Pfeifer, eds.), vol. 4850 of *Lecture Notes in Computer Science*, pp. 358–369, Springer Berlin Heidelberg, 2007.

[25] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[26] G. Torres, K. Jaime, and F. Ramos, "Brain architecture for visual object identification," *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, vol. 7, no. 1, pp. 75–97, 2013.

[27] G. R. F Bellifemine, A Poggi, "Jade – a fipa-compliant agent framework," *Proceedings of PAAM*, 1999.

[28] J. Garofalakis, Y. Panagis, E. Sakkopoulos, and A. Tsakalidis, "Web service discovery mechanisms: Looking for a needle in a haystack," in *International Workshop on Web Engineering*, vol. 38, 2004.

[29] F. I. C. A. Specification, "Foundation for intelligent physical agents, 2000."

[30] N. Carriero, D. Gelernter, and J. Leichter, "Distributed data structures in linda," in *Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pp. 236–242, ACM, 1986.

[31] P. A. Bernstein, "Transaction processing monitors," *Communications of the ACM*, vol. 33, no. 11, pp. 75–86, 1990.

[32] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311–327, 2004.

[33] A. K. J. Oliver, *Modelo de memoria declarativa para criaturas virtuales basado en neurociencias*. 2015.

[34] W. Beaton and J. d Rivieres, "Eclipse platform technical overview," *Retrieved on November*, vol. 2, p. 2009, 2006.

[35] O. M. F. R. G. T. Karina Jaime, Armando Cervantes and M. Ramos, "A middleware for integrating cognitive architectures," 2015.

# Apéndice A

# Documento de especificación de requerimientos de software

## A.1. Introducción

En este documento se especifican todos los requerimientos que fueron recolectados del equipo de investigación para el desarrollo del Middleware Kuayalotl.

Los requerimientos del se dividen en dos:

- Requerimientos de usuario: Es textualmente estipulado por el usuario imperando necesidades que desea ver cubiertas.
- Requerimientos de sistema: Son trasladados de los requerimientos del usuario, estos ya designan una característica o especificación concreta del sistema.

## A.2. Especificación de los requerimientos

### A.2.1. Requerimientos de Usuario

| ID | requerimiento |
|---|---|
| RUF001 | La ejecución de kuayolotl debe ser distribuida. |
| RUF002 | Deben de poder estar en ejecución más de un kuayolotl. |
| RUF003 | Tiene que ser sencillo de utilizar. |
| RUNF0044 | Debe de contar con ejemplos del uso del sistema y su documentación. |

There is more to come

| ID | requerimiento |
|---|---|
| RUF005 | Contemplar diferentes lenguajes de programación para la implementación de funciones. |
| RUF006 | Debe de ser un middleware por paso de mensajes. |
| RUF007 | Los mensajes del middleware tienen que ser flexibles. |
| RUF008 | Integrar funciones cognitivas desarrolladas con la metodología propuesta por el grupo. |
| RUF009 | El sistema ambiente será encargado de proporcionar estímulos al middleware. |
| RUF010 | El middleware debe ser capaz de comunicarse con un sistema ambiente. |
| RUF011 | Varias instancias del middleware pueden comunicarse con el mismo sistema ambiente. |
| RUF012 | Debe de ser multiplataforma. |
| RUF013 | El trabajo será realizado por pequeños programas llamados nodos. |
| RUF014 | Existirán nodos pequeños y nodos grandes. |
| RUF015 | Los nodos grandes deben de funcionar como ruteadores. |
| RUF016 | Los nodos grandes deben de cargar su configuración de manera dinámica. |
| RUF017 | Los nodos pequeños deben encargarse de procesar las señales. |
| RUF018 | Los nodos pequeños deben ser capaces de enviar señales. |
| RUF019 | Los nodos grandes deben poder hacer un multicast solo para sus nodos pequeños. |
| RUF020 | Los nodos grandes deben poder iniciar nodos pequeños. |
| RUF021 | Los nodos pequeños del mismo nodo grande tienen que poder comunicarse entre ellos. |
| RUF022 | Un nodo grande debe permitir crear más de un solo tipo de nodos pequeños. |
| RUF023 | Los nodos utilizados para la memoria de trabajo deben tener la característica de mantenerse en ejecución durante todo el tiempo que sean requeridos. |
| RUF024 | Debe de contar con opciones para la creación de un Log (tiempo de inicio, tiempo final, duración, nodo que solicita el proceso, proceso realizado, etc.). |
| RUF025 | Un nodo grande debe poder enviar un mensaje a todos sus nodos pequeños. |
| RUF026 | Puede existir una jerarquía en los nodos grandes. |
| RUF027 | Los mensajes del middleware serán Spikes. |
| RUF028 | Los mensajes del middleware deben de ser lo más ligeros posible. |
| RUF029 | El proceso de actualización de tablas de direcciones debe de ser eficiente (utilizar pocos mensajes). |
| RUF030 | Un ID único identificará a cada nodo (nodo grande, nodo pequeño), los nodos de intersección de area, compartirán ID. |
| RUF031 | Este ID debe de ser lo más reducido posible. |
| RUF032 | Los paquetes que lleguen a los nodos grandes deben de ser procesados paralelamente. |
| RUF033 | Los paquetes de datos deben ser auto-contenidos. |

There is more to come

| ID | requerimiento |
|---|---|
| RUF034 | Un nodo grande debe de tener la opción de manejar sus nodos pequeños por medio de un mapeo (cuenten con un índice). |
| RUF035 | Cada pc solo deberá contar con una dirección multicast en el router. |
| RUF036 | El usuario final no tendrá que crear clases o métodos adicionales para correr el middleware. Ej, crear un main que cree instancias de varias clases con un main. |

## A.2.2. Requerimientos de Sistema

| ID | ID | requerimiento |
|---|---|---|
| RS001 | RUF002 | El middleware debe de permitir la ejecución simultánea de varias entidades. |
| RS002 | RUNF004 | El middleware debe de ser entregado con su documentación. |
| RS003 | RUNF004 | El middleware debe de ser entregado con ejemplos correspondientes. |
| RS004 | RUF001 | El middleware permitirá de manera transparente la ejecución distribuida en un ambiente local. |
| RS005 | RUF005 | El middleware debe de ser multilenguaje. |
| RS006 | RUF003 | Pueden correr varios nodos grandes en el mismo equipo. |
| RS007 | RUF003 | La intersección de funcionalidades de nodos grandes con el mismo nombre debe de ser transparente. |
| RS008 | RUF015 | La función de los nodos grandes debe de ser encaminar la información a los nodos pequeños. |
| RS009 | RUF015 | Los nodos grandes obtendrán las direcciones de otros nodos grandes conforme se van registrando. |
| RS010 | RUF029 | El registro de los nodos grandes se debe realizar por multicast. |
| RS011 | RUF029 | La solicitud de dirección de otro nodo se debe realizar por multicast. |
| RS012 | RUF015 | Los nodos grandes pueden solicitar la dirección de otro nodos grande. |
| RS013 | RUF022 | Un mismo nodo grande podrá manejar varios tipos de nodos pequeños. |
| RS014 | RUF021 | Un nodo pequeños podrá enviar información directamente a otro nodo pequeño del mismo nodo grande. |
| RS015 | RUF015 | Los nodos grandes deben cargar su información de ruteo de manera dinámica por medio de un archivo de texto. |
| RS016 | RUF025 | Un nodo grande debe tener la opción de crear un grupo multicast exclusivo para sus nodos pequeños. |
| RS017 | RUF023 | Se debe contar con un sistema de backup en el cual se guarda el estado actual de la ejecución de un nodo para posteriormente cargarse. |

There is more to come

| ID | ID | requerimiento |
|---|---|---|
| RS018 | RUF023 | Los nodos deben de poder configurarse como persistentes o momentáneos. |
| RS019 | RUF024 | Los nodos grandes deben de contar con la opción de guardar un log de los eventos que han sucedido (tiempo de inicio, tiempo final, duración, nodo que solicita el proceso, proceso realizado, etc.). |
| RS020 | RUF002 | Debe de poderse correr nodos grandes pertenecientes a diferentes entidades sobre el mismo equipo. |
| RS021 | RUF026 | Debe existir una jerarquía de nodos grandes, la cual permitirá tener niveles en el ruteo de los paquetes. |
| RS022 | RUF029 | Cuando un nodo grande sea inicializado este enviara un mensaje multicast, para informar de su existencia. Este mensaje debe de ser contestado por los demás nodos enviando sus datos de ruteo. |
| RS023 | RUF029 | Cuando un nodo grande requiera la dirección de un nodo que no conoce enviará pro multicast, un mensaje de búsqueda, que será contestado con un mensaje de encontrado. |
| RS024 | RUF030 | El ID único de cada nodo, debe de ser asignado por el usuario en caso de los nodos grandes y los nodos grandes asignan automáticamente el ID de sus nodos pequeños. |
| RS025 | RUF030 | El ID incluirá los campos para identificar Nodo grande, tipo de nodo pequeño, nodo pequeño. |
| RS026 | RUF030 | El ID estará dividido en: Área: 12 bits, tipo: 5 bits, Index: 15 bits |
| RS027 | RUF031 | El ID no deberá exceder los 4 bytes de longitud. |
| RS028 | RUF032 | Cuando un paquete sea recibido por un nodo grande, este creara un hilo para el manejo de este mensaje. |
| RS029 | RUF021 | El proceso por el que un nodo pequeño obtiene la dirección de algun hermano, debe ser una solicitud al nodo padre, contestada con la dirección de un hermano. |
| RS030 | RUF024 | Cada nodo (grande o pequeño), tendrá su archivo log individual. |
| RS031 | RUF024 | El nombre de una archivo log esta constituido por la fecha año-mes-dia-hora-minuto-segundo mas el ID de un nodo. |
| RS032 | RUF024 | Para facilitar su lectura, los archivos log estarán escritos en XML. |
| RS033 | RUF024 | Los archivos log, tendrán la opción de incluir o no, la información de eventos en los nodos (creación de un nodo pequeño, arribo de mensajes de diferentes tipos). |
| RS034 | RUF029 | Cuando un nodo grande ingrese al grupo multicast, mandara un mensaje SINGINAREA, el cual al contendrá el ID, IP y Puerto del nodo que ingresa. |

There is more to come

| ID | ID | requerimiento |
|---|---|---|
| RS035 | RUF029 | Cuando un nodo grande reciba un SINGINAREA, este agregara al nodo entrante a su tabla de ruteo y contestara con un mensaje HANDSHAKE, el cual contendrá el ID, IP y Puerto del nodo que recibió el SINGINAREA. |
| RS036 | RUF029 | Al recibir un HANDSHAKE el nodo agrega el emisor a su tabla de ruteo y envía los mensajes de DATA pendientes en caso de existir. |
| RS037 | RUF033 | Un paquete de DATA, debe de incluir qué área lo envió, que tipo de dato es y los datos. |
| RS038 | RUF020 | Cuando un nodo grande recibe un paquete DATA este debe ser redireccionado a algún hijo. En caso de ser necesario se crearán más hijos. |
| RS039 | RUF020 | Cuando se crea un nodo hijo, este inicia un nuevo proceso, el cual recibe como parámetros, su ID, el ID y dirección del padre. |
| RS040 | RUF020 | Al iniciar el nodo hijo agrega a su tabla la dirección del padre y le manda un mensaje SINGINCHILD. |
| RS041 | RUF020 | Cuando un nodo grande recibe un mensaje SINGINCHILD, éste lo agrega a su tabla de nodos pequeños y en caso de ser necesario envía mensajes DATA pendientes. |
| RS042 | RUF029 | Los nodos pequeños, contarán con una tabla de incluirá información de diferentes áreas, así como otra para sus hermanos (nodos de diferente tipo, del mismo nodo grande). |
| RS042 | RUF029 | En caso de que un nodo pequeño no cuente con la información de un área, almacenará el mensaje en una lista de espera. luego este enviara un mensaje SEARCHNODE, a su nodo padre. |
| RS043 | RUF029 | Un nodo padre al recibir un mensaje SEARCHNODE, verificará si tiene la información solicitada, en caso de tenerla contestara al nodo solicitante con un mensaje FINDNODE, en caso de no tenerla, almacenará la solicitud SEARCHNODE, en una lista de espera, luego enviará un mensaje SEARCHMULTICAST, para buscar el área solicitada. |
| RS044 | RUF029 | Cuando un nodo grande recibe un mensaje SEARCHMULTICAST revisará si el es el área buscada, en caso de ser lo enviará su información por medio de un mensaje FINDNODE. |
| RS045 | RUF029 | Cuando un nodo Grande recibe un mensaje FINDNODE, lo agrega a su tabla de ruteo y posteriormente envía los mensajes pendientes para este nodo. En caso de que este nodo haya sido solicitado previamente reenviara el mensaje FINDNODE. |
| RS046 | RUF029 | Cuando un nodo pequeño recibe un mensaje FINDNODE, lo agregara a su tabla de ruteo y posteriormente envía los mensajes pendientes para este nodo. |

There is more to come

| ID | ID | requerimiento |
|---|---|---|
| RS047 | RUF026 | Cuando un nodo grande crea a un nodo grande como su hijo, el hijo no contará con funciones multicast. |
| RS048 | RUF026 | Un nodo grande de tipo hijo, puede crear sus propios hijos (nodos grandes o nodos chicos). |
| RS049 | RUF026 | Cuando un nodo grande tipo hijo requiere información de un área, lo actualizara por el mismo proceso que lo hace un nudo pequeño. |
| RS050 | RUF026 | El nodo padre grande, no hará distinción entre el tipo de hijo (nodo grande o nodo chico). |
| RS051 | RUF022 | El manejo de nodos hijos, será un módulo del nodo padre (por cuestiones de simplicidad). |
| RS052 | RUF022 | El módulo de manejo de nodos hijos, se encargara de la creacion, y gestion y distribucion de paquetes a los nodos pequeños. |
| RS053 | RUF022 | El módulo de manejo de nodos hijos, contendrá los diferentes tipos de hijos y la diferencia en la administración de cada uno. |
| RS054 | RUF033 | El módulo de manejo de nodos hijos contará con una opción para administrarlos por medio de una hashtable que relacione un índice con un nodo hijo. |
| RS055 | RUF035 | Cada equipo contará con un nodo de interfaz que reciba y direcciona solo el tráfico proveniente de multicast. |
| RS056 | RUF036 | El nodo interfaz también se encarga de inicializar los nodos grandes y pequeños y en caso de ser necesario por balance de carga enviarlo a otro equipo. |

# Apéndice B

# Documento de diseño de Software

## B.1. Introdución

te documento contiene información detallada acerca del plan y especificaciones de diseño para el middleware, empezando por sus estructuras generales y avanzando hacia los detalles del funcionamiento.

Los diagramas de este documento están basadas en la especificación UML 2.5

### B.1.1. Público objetivo

Este documento esta orientado a personas con conocimiento tecnico sobre el software y su diseño, principalmente a desarrolladores de software y lideres de proyecto.

### B.1.2. Definiciones, acrónimos y abreviaturas

- Middleware: Es una capa de abstraccion de software la cual cobree comunicacion transparente entre dos componentes de un sistema.
- Entidad: En este documento la palabra entidad es manejada como sinónimo de entidad virtual. Una criatura que habita en un ambiente virtual y exive un comportamiento.
- Multicast: Es un protocolo de comunicacion en el cual el route crea una lista de subscriptures bajo peticion y el pateque que llegue al grupo sera reenviado a todos los subscriptures.

### B.1.3. Referencias

## B.2. Estructura general

La unidad principal del middleware son los nodos grandes, estos tienen como objetivo encaminar la información hacia su destino: los nodos pequeños, tomando además en consideración, como y cuando los nodos pequeños deben de generarse para eficientizar el uso de recursos.

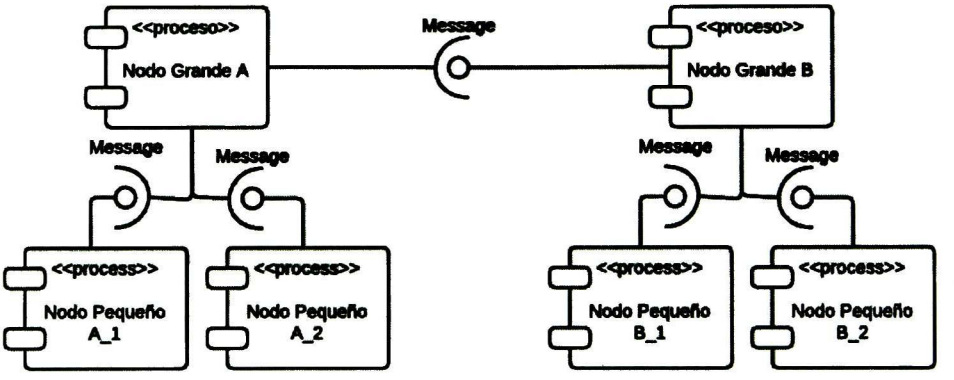



Figura B.1: Estructura general de la intercomunicación de los modulos.

### B.2.1. Inicialización de un Nodo Grande

Al inicializarse el nodo grande debe, conseguir un puerto libre, suscribirse a un grupo multicast, y activar 2 puertos de escucha, una para las conexiones p2p y otro para los mensajes multicast. se mandan x paquetes los titulos para México y los legalicen

Una vez que se a levantado estos puertos de escucha, envía un mensaje multicast, llamado SINGIN, en el cual envía su ID, IP y puerto, para que los demás nodos grandes lo agregue a su lista y envien su informacion a traves de un HANDSHAKE.

De esta manera las tablas de ruteo entre nodos grandes se mantienen actualizadas.

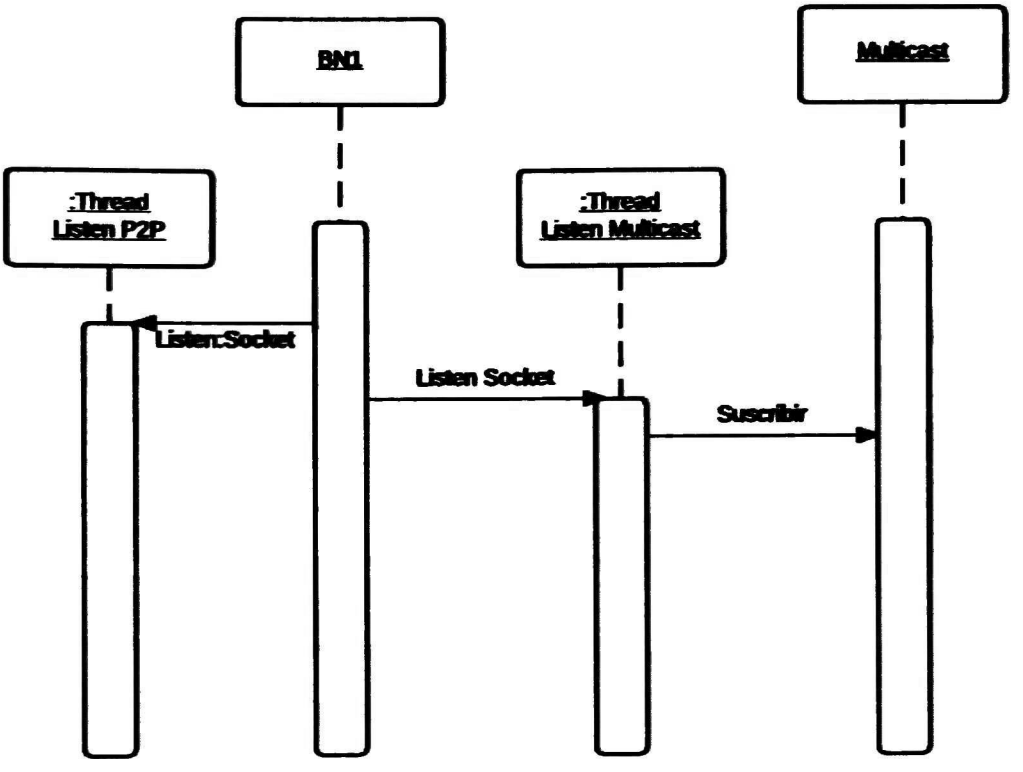



Figura B.2: Diagrama de interaccion entre BigNodes y SmallNodes.

## B.2.2. Inicialización de los Nodos pequeños

Los nodos pequeños son inicializados por medio de comando al SO, ya que se ocupa iniciar un proceso nuevo, además de que esto da independencia de lenguaje y equipo (ya que el comando puede ser administrado por otro equipo).

Al ser iniciados, los nodos pequeños, levantan un hilo de escucha P2P, y envían sus datos a su big Node creador, la información de su localización se encuentra en los parámetros recibidos

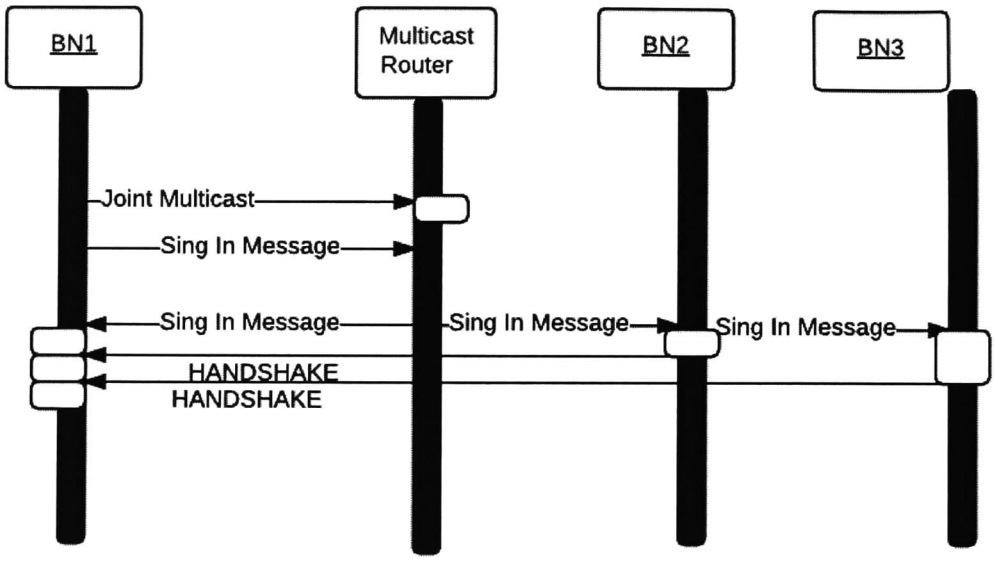



Figura B.3: Diagrama de interaccion del proceso de descubrimiento y handshake de un Big-Node.

### B.2.2.1. Sing In Area Protocol

Cuando un nodo grande es inicializado se subscribe al grupo multicast e inmediatamente despues, se presenta con los otros nodos con un SIA message, el cual contiene la dirección P2P del nodo. Los cuales le contestan con un Handshake message, el cual contiene la direccion P2P del nodo.

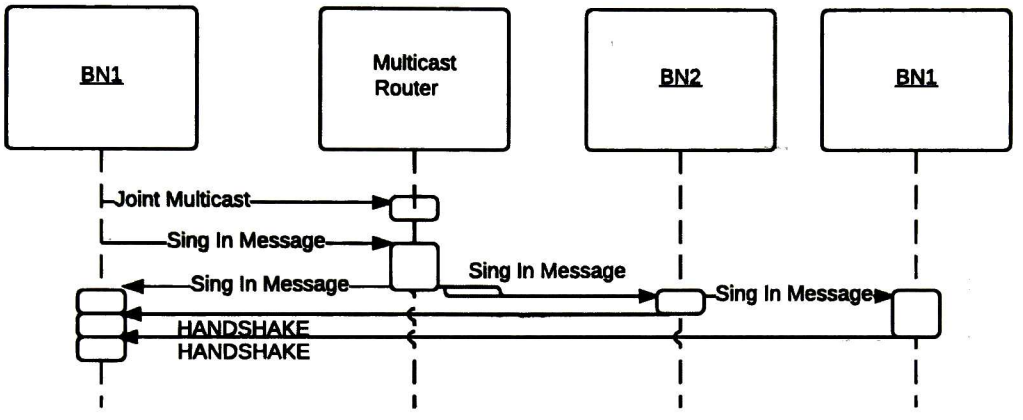



Figura B.4: Diagrama de seciencia del protocolo SIA.

#### B.2.2.2. Sing In Child Protocol

Cuando el BN recive un mensaje de datos este puede ser enviado a un tipo de hijo. Al momento de enviarlo, puede ser posible que no existan hijos libres disponibles para poder procesar el mensaje. Por esto es nesario iniciar un nodo.

Cuando un nodo pequeño es inicializado, este recive como argumento la dirección IP y el peurto del nodo padre. Una vez que el padre recive el SIC message, el cuan contiene la dirección IP y puerto del hijo, lo agrega a su lista de hijos libres, entonces procede a enviarle un mensaje de la cola de mensajes pendientes.

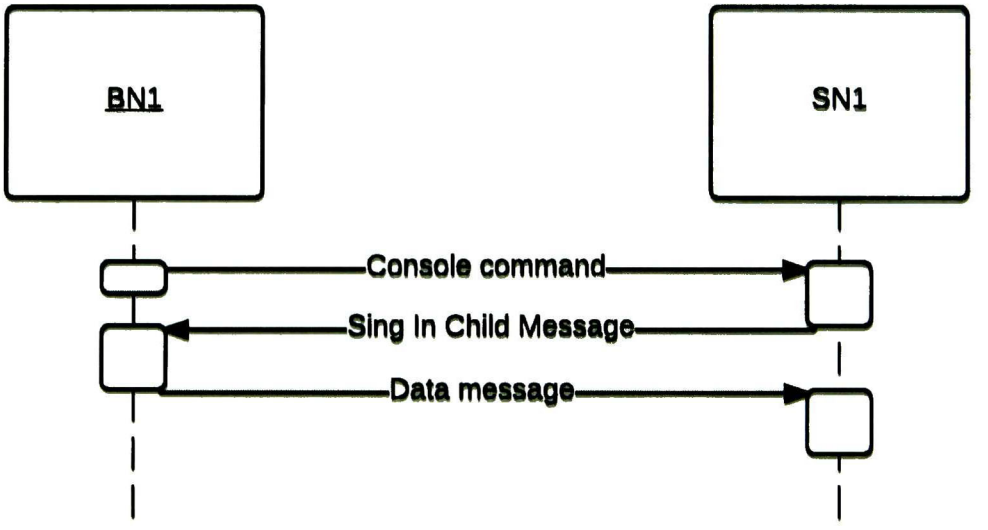



Figura B.5: Diagrama de sequencia del protocolo SIC.

### B.2.2.3. Search Node Request

En caso de que un SN tenga que enviar un mensaje de datos a a un BN y este no conozca su dirección, este tiene que actualizar su informacion de ruteo, por lo tanto preguntara su BN la dirección que el SN no conoce a traves de un Search Node Request.

En caso de conocerlo como se observa en la figura 3.5 este le enfia un mensaje FindNode, el cual contiene la dirección IP y el puerto del nodo que esta solicitando, es entonces que el SN procede a enviar la información.

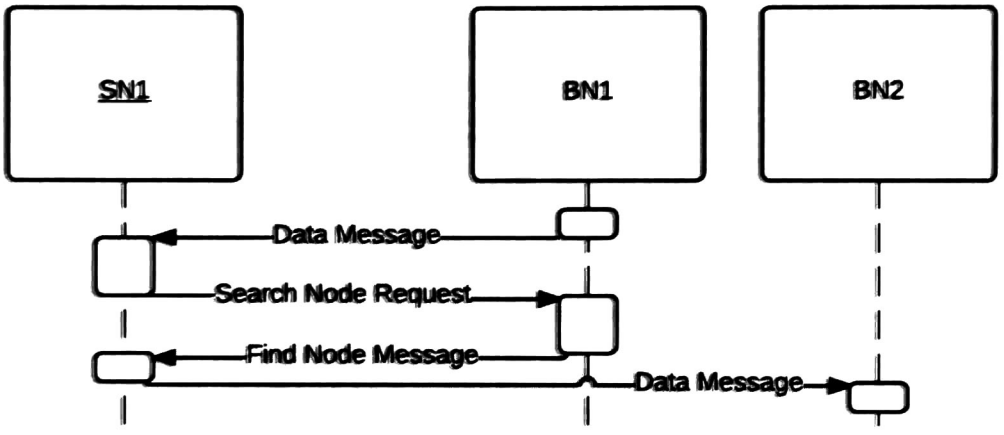



Figura B.6: Diagrama de sequencia del protocolo SNR, el BN conoce la dirección que el SN solicita.

## B.2.3. Diagramas de clases

### B.2.3.1. Class BigNode y su estructura

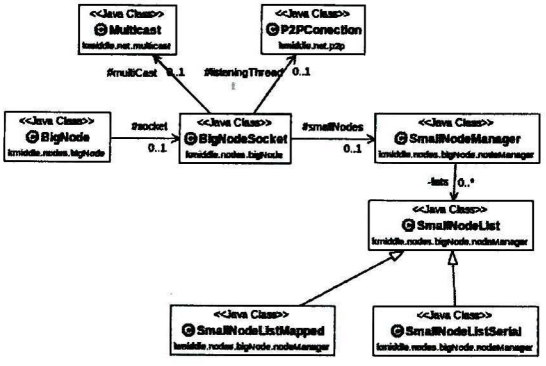



Figura B.7

### B.2.3.2. Class BigNode

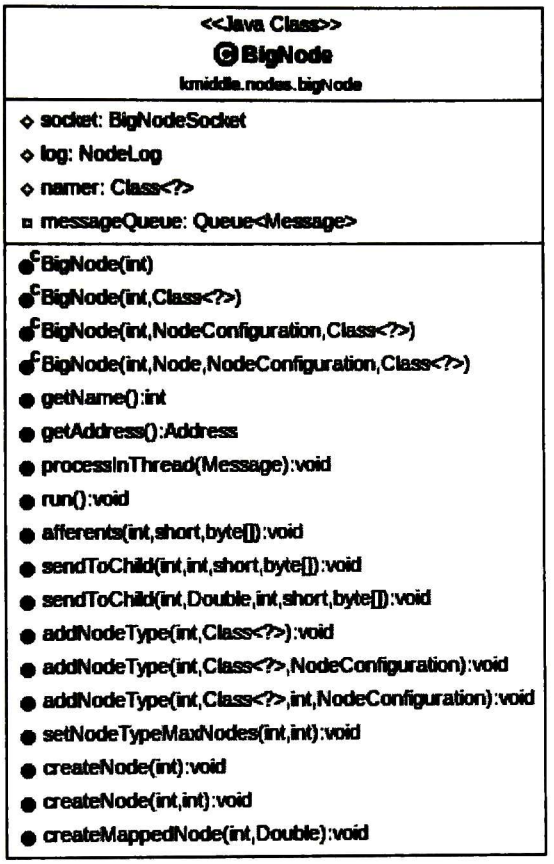



Figura B.8

### B.2.3.3. Class BigNodeSocket

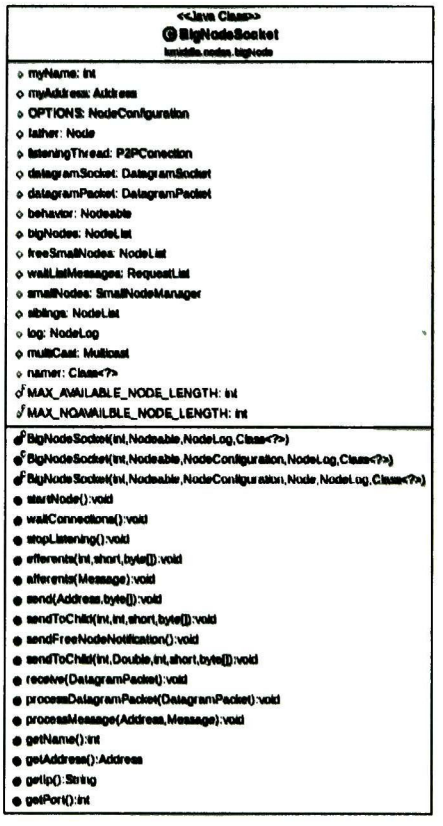



Figura B.9

#### B.2.3.4. Class Multicast

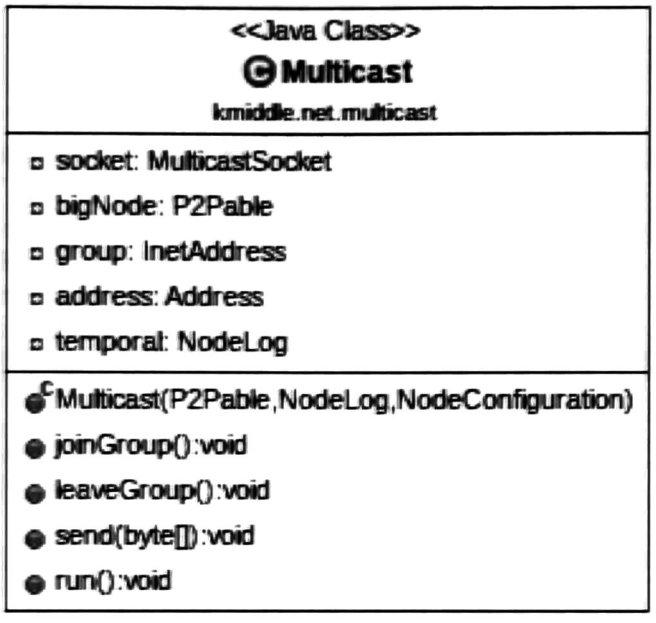



Figura B.10

### B.2.3.5. Class P2PConection

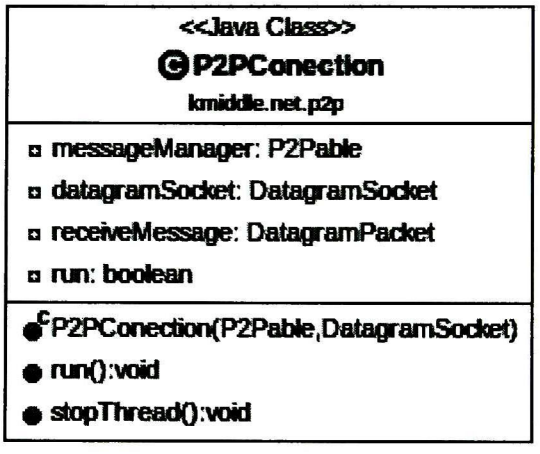



Figura B.11

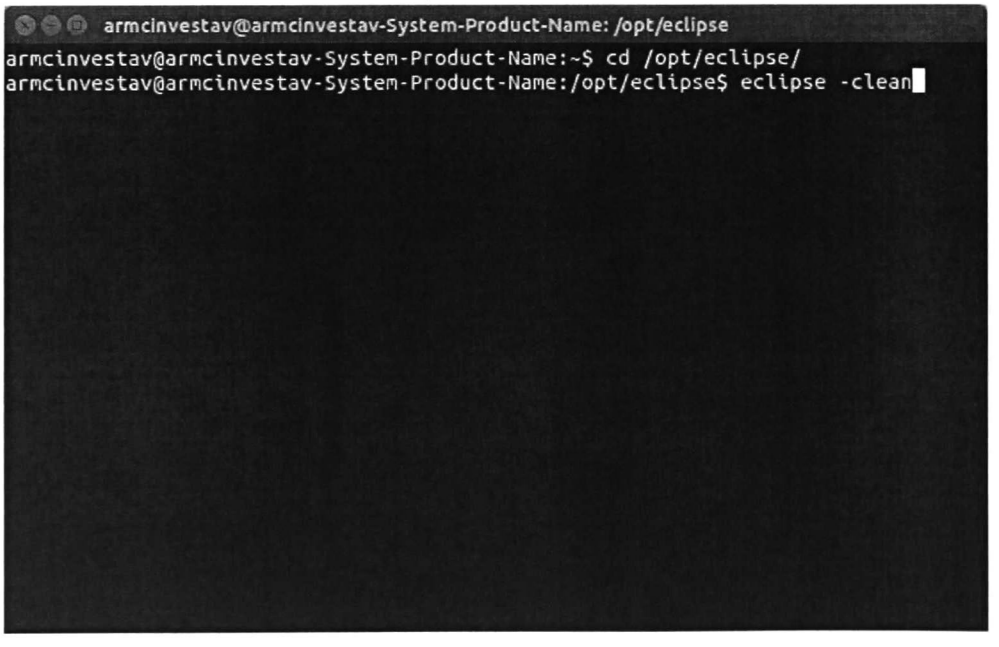
# Manual de Instalación Middleware y Plugin Kuayolotl

## Procedimiento para la instalación del Plugin

1.- Ir a la carpeta en la que está instalado Eclipse
2.- Entrar a la carpeta **dropins**
3.- Crear la carpeta **plugins**


4.- **Dentro de plugins** copiar el archivo **KMiddlePlugin...**

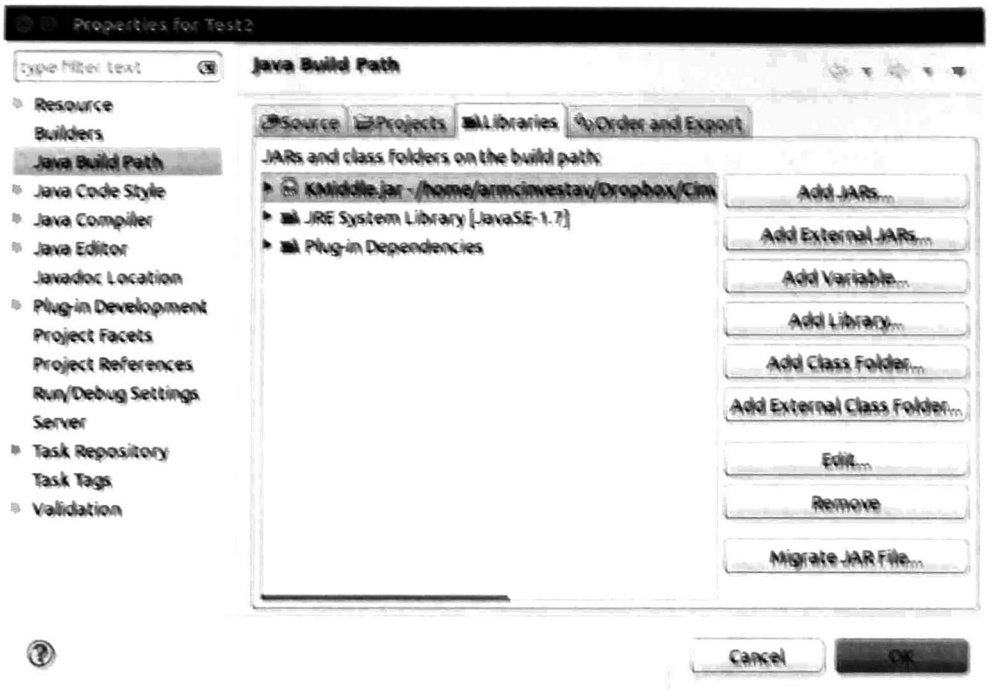

5.- Desde consola iniciar el IDE con la opcion **-clean**




Con estos paso el plugin está instalado.
Nota: No es necesario abrir el ide desde consola después de haber terminado de instalar.

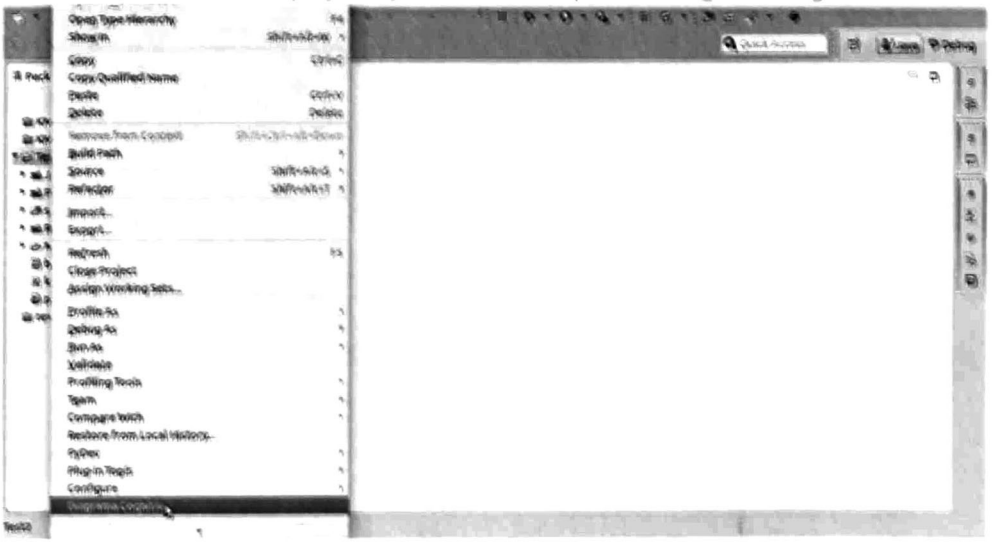
# Usando el Plugin

1.- Se crea un nuevo proyecto en java
2.- Se agrega la librería KMiddle.jar

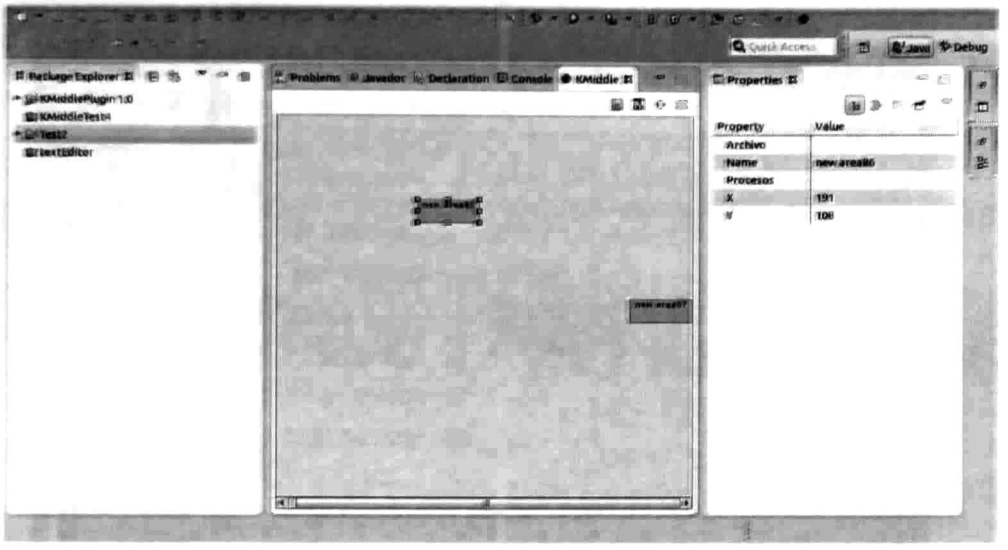



3.- Ahora podemos Abrir nuestro diagrama cognitivo:

Click derecho en el proyecto y en las últimas opciones **Diagrama Cognitivo**



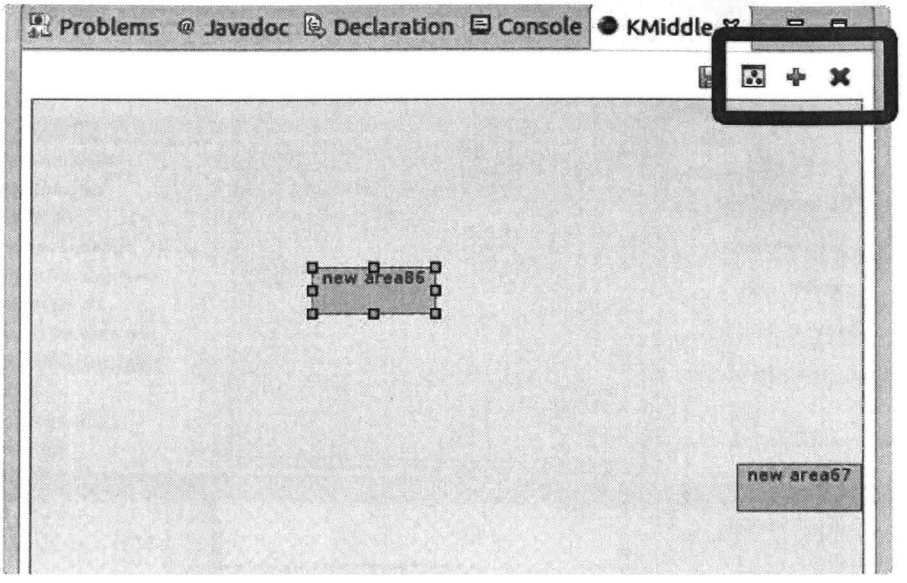
Se verá esta pantalla



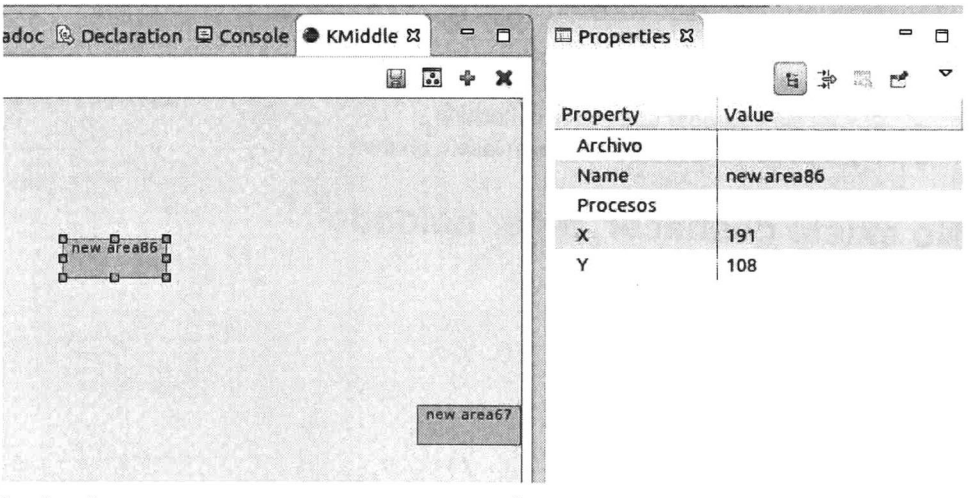

En la barra de herramientas vemos 3 botones (ignorar el de guardar)

- El icono rectangular es para agregar una nueva área
- El + es para agregar una nueva coneccion
- Y la X es para eliminar ya sea un enlace o un área

## No existe deshacer, tener cuidado

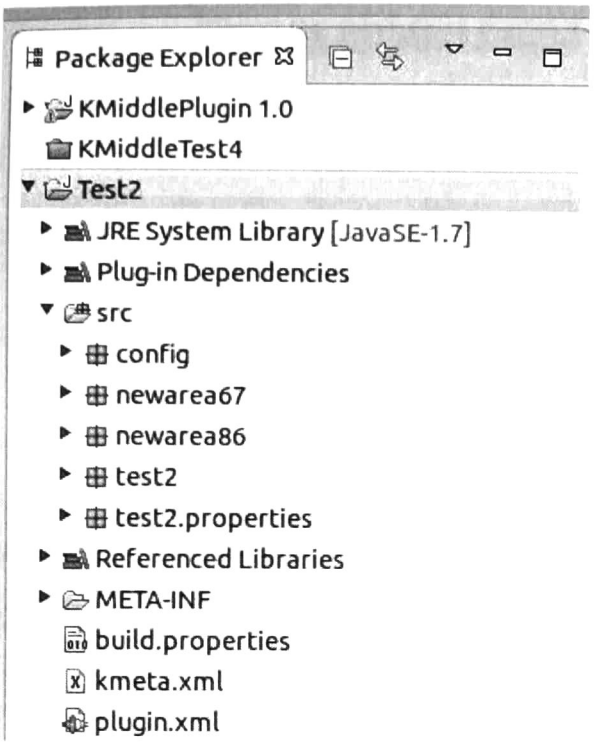



Al hacerle click a un area podemos ver sus propiedades




Aquí podemos agregar nuevos procesos o acceder directamente al código fuente.

Para ver los cambios en el código es necesario actualizar el proyecto **F5**


Package Explorer
KMiddlePlugin 1.0
KMiddleTest4
Test2
JRE System Library [JavaSE-1.7]
Plug-in Dependencies
src
config
newarea67
newarea86
test2
test2.properties
Referenced Libraries
META-INF
build.properties
kmeta.xml
plugin.xml

# CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
# UNIDAD GUADALAJARA

**El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis**

**Entorno de desarrollo para arquitecturas cognitivas**

**del (la) C.**

**Armando CERVANTES HERNÁNDEZ**

**el día 21 de Agosto de 2015.**

**Dr. Félix Francisco Ramos Corchado**
**Investigador CINVESTAV 3C**
**CINVESTAV Unidad Guadalajara**

**Dr. Luis Ernesto López Mellado**
**Investigador CINVESTAV 3C**
**CINVESTAV Unidad Guadalajara**

**Dr. Pablo Moreno Villalobos**
**Investigador CINVESTAV 3C**
**CINVESTAV Unidad Guadalajara**

**Dr. Juan Manuel Ramírez Arredondo**
**Investigador CINVESTAV 3C**
**CINVESTAV Unidad Guadalajara**

Dr. Gustavo Alejandro Torres Blanco
Profesor Investigador
Universidad Autónoma de Guadalajara