Universidad de las Ciencias Informáticas

Facultad 5

Título: Propuesta de estrategia para la toma de decisiones en el subsistema de balance de carga del Módulo de Almacenamiento y Gestión de Datos de Históricos del SCADA *Guardián del Alba*.

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

Autor:

Lisandra Caridad Martínez Nieto

Tutores:

Ing. Abdelaziz de la Horra Díaz

Ing. Lazaro Abreu Reche

Ciudad de la Habana, 2010

Declaración de autoría

DECLARACION DE AUTORÍA

Declaramos ser autores de este trabajo y reconocemos a la Universidad de las Ciencias Informáticas los derechos primordiales del mismo con carácter exclusivo.

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

Ing. Lazaro Abreu Reche

_____

FIRMA DEL TUTOR

Ing. Abdelaziz de la Horra

_____

FIRMA DEL TUTOR

Lisandra Caridad Martínez Nieto

_____

FIRMA DE LA AUTORA

Agradecimientos

A Lazaro por el tiempo que me dedicó.

A Lety, el Ferna y el Ferny por ser más que mis amigos y ayudarme siempre en todo.

A todos mis amigos, en especial a Yanesita.

Dedicatoria

A mi madre por creer siempre en este momento. A Lazaro y a Luli los dos amores de mi vida.

Resumen

El módulo de Almacenamiento y Gestión de Datos Históricos recibe a través del Middleware (módulo encargado de transporte de datos) un elevado volumen de datos que debe procesar. Estas tareas pueden en un determinado momento salirse de control por una falla en el sistema que daría origen a una avalancha de datos. Esto puede ocasionar un retraso en el procesamiento y almacenamiento de los mismos haciendo que se congestione el módulo y cause pérdida de estos datos. Por esto se hace necesario agregar al módulo un subsistema de balance de carga que le permitiera distribuir el trabajo en esos casos. Una mejor distribución de la carga en el módulo puede traer la menor pérdida posible de los datos en situaciones anormales.

Para ello la investigación propone una nueva estrategia de balanceo, que utiliza la lógica difusa como algoritmo de toma de decisiones, que optimiza el funcionamiento del subsistema de balanceo. Además de una serie de herramientas totalmente libres que dieron lugar a una herramienta de desarrollo y evaluación para realizar pruebas para la comparación entre la estrategia propuesta y las ya implementadas en el subsistema de balance de carga.

Palabras Claves

Balance de carga, Estrategia, Lógica Difusa, Módulo de Almacenamiento y Gestión de Datos Históricos, Toma de Decisiones.

Índice

Índice

ÍNDICE DE FIGURAS.

INDICE DE TABLAS.

Introducción

Cuando uno se encuentra ante un problema, definido por un estado inicial, un estado final deseado, una variedad de posibles acciones que emprender, y un entorno sobre el que se ejercen estas acciones, se está ante un problema de decisión. Este problema consiste en decidir qué acciones emprender, de entre las posibles acciones alternativas, y en qué orden, para conseguir el resultado deseado. Los problemas de decisión surgen de manera continua en la vida cotidiana. El determinar por qué camino volver a casa, cómo invertir los ahorros, o cómo modificar la temperatura del hornillo en el que hierve el agua de los macarrones, se reduce a resolver un problema de decisión [1].

La toma de decisión constituye hoy, sin duda, uno de los campos de aplicación más fértiles de las técnicas de la Inteligencia Artificial (IA). La amplia aplicabilidad de estas técnicas reside en el hecho de que una gran cantidad de problemas (control de sistemas dinámicos, predicción, optimización, clasificación, planificación, búsqueda heurística) se reduce, o conlleva, a un problema de decisión. El estudio de los algoritmos de toma de decisión es por tanto especialmente interesante, pues promete grandes posibilidades de aplicación, y satisface una necesidad creciente de sistemas de apoyo a la hora de resolver muchos problemas.

En la Universidad de las Ciencias Informáticas (UCI), específicamente en la Facultad 5, se desarrolla un Sistema de Control y Adquisición de Datos (SCADA), que tiene como objetivo el control de procesos en la industria. Este sistema está concebido con una arquitectura distribuida donde cada módulo realiza una función específica. Uno de estos módulos es el de Almacenamiento y Gestión de Datos Históricos (AGDH). Este recibe, a través del módulo encargado del transporte de datos (*Middleware*), un cuantioso volumen de información que debe procesar para posteriormente almacenarla en la Base de Datos (BD) como datos históricos. Además, debe proveer a otros módulos mecanismos de acceso a estos datos.

En determinados momentos el sistema - por condiciones desfavorables o descuido de los operarios- puede alterar su funcionamiento normal, generando volúmenes de datos muy superiores a los esperados por concepto de eventos normales o críticos (alarmas). Este suceso, conocido como avalancha, puede

congestionar el módulo de AGDH al impedir que procese un volumen de información que supera los requerimientos planteados para el módulo. En tales casos, el resultado es siempre la pérdida de datos.

Para darle solución a esta situación se incorporó un subsistema de balance de carga que maneja varios hilos de procesamiento de la información recibida, ganando en la capacidad de soportar las avalanchas anormales de datos. Este subsistema cuenta con estrategias para determinar cuántas réplicas (hilos de procesamiento) debe utilizar en cada momento y cuál de ellas es la encargada de asumir un determinado volumen de información previamente recibida. Hoy este módulo cuenta con 3 estrategias basadas en algoritmos aleatorios y cíclicos que no tienen en cuenta determinados elementos: tipos de estructuras a procesar, complejidad para procesar y almacenar cada estructura, tamaño de los buffers de inserción para las variables, etc. que influyen directamente en una mejor distribución del trabajo entre las réplicas.

De esta situación surge como **Problema Científico** a resolver: ¿Cómo lograr una mejor distribución de la carga en el subsistema de balance de carga del módulo de AGDH del SCADA *Guardián del Alba*? Para darle solución al problema en cuestión, este trabajo tiene como **Objeto de Investigación** la Inteligencia Artificial para la toma de decisiones, concentrándose específicamente en los algoritmos de toma decisiones como **Campo de Acción**.

El **Objetivo General** de la investigación es: Proponer una estrategia de toma de decisiones para el subsistema de balance de carga del módulo de AGDH que optimice la distribución de la carga entre los hilos de procesamiento.

Para dar cumplimento a este objetivo se plantean las siguientes **Tareas de Investigación**:

1. Estudio de diferentes patrones de software aplicables a la toma de decisiones en aplicaciones de balance de carga para seleccionar el que se aproxime a una posible solución del problema.
2. Estudio de los diferentes algoritmos de toma de decisiones existentes aplicables al balance de carga para seleccionar el que por sus características se ajuste mejor a las necesidades del problema.
3. Estudio de la arquitectura del subsistema de balance de carga del módulo de AGDH del SCADA *Guardián del Alba* para llevar a cabo el diseño de la estrategia.

4. Implementación de la estrategia propuesta que incluya el algoritmo de toma de decisiones seleccionado.
5. Realización de pruebas para comparar el funcionamiento de la estrategia propuesta con las ya implementadas para el subsistema de balance de carga.

La investigación está estructurada en tres capítulos. En el primero de ellos, dedicado a la Fundamentación Teórica, se explican las particularidades del módulo de AGDH del SCADA *Guardián del Alba*, así como los principales conceptos utilizados durante la investigación. Además, se caracterizan los algoritmos de toma de decisión y se aborda en general el estado del arte de nuestro objeto de estudio.

En el capítulo dos: Solución Propuesta, se plantean las herramientas y metodologías que posibilitan el desarrollo de la misma así como las reglas y requerimientos que debe seguir.

En el tercer capítulo se realiza el Análisis de los Resultados. Aquí se exponen los cambios necesarios en las clases que controlan el balance de carga para poder acoplar la nueva estrategia y el nuevo diseño de las clases una vez incorporada esta. Además, se explica el funcionamiento y se aplican pruebas para probar la validez de la solución propuesta.

A continuación, en las conclusiones se hace una generalización en referencia al nuevo conocimiento que generó el presente trabajo a través del cumplimiento del objetivo general y tareas de la investigación.

En las recomendaciones se ponen de manifiesto las posibilidades de aplicación de los resultados y se sugieren mejoras para optimizar el rendimiento de la solución propuesta.

Por último, en los anexos se muestran imágenes y gráficas que complementan el contenido de la investigación y contribuyen a la comprensión de sus resultados.

Diseño Metodológico.

La palabra método procede del griego *méthodos* y quiere decir vía de investigación del conocimiento, teoría y estudio. La forma en que el sujeto se aproxima al objeto en la investigación responde a dos niveles en el conocimiento: el teórico y el empírico, por tanto, los métodos que lo permiten serán teóricos o empíricos.

Los primeros permiten explicar y predecir fenómenos objetivos. Ayudan a la interpretación de los datos empíricos encontrados y a la sistematización y generalización de los conocimientos.

Los empíricos, por su parte, se asocian a los procedimientos por los cuales se obtiene la información necesaria, directamente de la realidad. Sirven de apoyo y ayudan a enriquecer las valoraciones teóricas. Se asocian a los momentos de la investigación en que la interacción del sujeto con el objeto de investigación es directa con la realidad y al reflejo obtenido de esas propiedades y cualidades.

La integración de ambos conduce a conclusiones más adecuadas en correspondencia con el objeto y con el objetivo de la investigación, por lo que tanto el método teórico como el empírico fueron de gran utilidad para el desarrollo del presente trabajo.

Métodos Teóricos

- Análisis histórico – lógico.
- Analítico - sintético.
- Inductivo – deductivo.

Análisis histórico-Lógico: Este procedimiento permite estudiar de forma analítica los fenómenos desde el punto de vista de su trayectoria histórica real. El empleo de este método en el curso de una investigación permite comprobar teóricamente cómo ha evolucionado un determinado fenómeno en un período de tiempo, en toda su trayectoria o en un fragmento temporal de la lógica de su desarrollo.

En el presente trabajo, como en toda investigación, es fundamental analizar el estado del arte del fenómeno objeto de estudio. A tal efecto se utiliza el análisis histórico-lógico para estudiar la evolución histórica de los algoritmos de toma de decisiones así como para evaluar su estado actual. También como parte del proceso investigativo, se ilustró el desarrollo del módulo AGDH del sistema SCADA y en particular, el progreso en el uso de los algoritmos. De igual modo se hizo un análisis de la evolución de C++ como lenguaje de programación con el fin de evaluar las características que permitieron identificarlo como el más adecuado para proponer nuestra solución.

Analítico - sintético: Este método permite analizar la esencia de los fenómenos a partir de los rasgos que los caracterizan y distinguen. Su objetivo es analizar las teorías, documentos y demás motivos de investigación, a través de la síntesis de los elementos más importantes que se relacionan con el objeto de estudio.

En el caso particular de la presente investigación, este método resultó de gran utilidad para determinar las principales características de los diferentes algoritmos de toma de decisiones que fueron analizados para establecer cuál se utilizaría para elaborar nuestra propuesta de estrategia. Otra aplicación de este método teórico en nuestro trabajo fue el estudio de los patrones de software en el servicio de balanceo a fin de juzgar su comportamiento. También se aplica el método analítico – sintético a la hora de analizar la arquitectura del subsistema de balance de carga, para entender su estructura y encauzar el desarrollo de una solución aplicable al mismo, y en el estudio general de todas las herramientas y tecnologías implicadas en nuestra investigación.

Inductivo-Deductivo: Como método teórico permite llegar a un grupo de conocimientos generalizadores, tanto desde el análisis de lo particular a lo general, como de lo general a lo particular.

Este procedimiento fue ampliamente utilizado en el transcurso de nuestra investigación:

- Al analizar a fondo el módulo de AGDH en general y los algoritmos de toma de decisiones en particular para determinar el problema científico.
- Al estudiar los algoritmos de toma de decisiones en general y los de lógica difusa en particular.
- En el análisis de las características particulares de C++ para finalmente evaluarlo en sentido general como lenguaje de programación.

- En la propia concepción de la solución propuesta a partir de cada uno de los elementos particulares que la componen.

Métodos Empíricos

Observación: Es el método más general que existe, ya que está presente en todas las acciones investigativas que se intenten, de una u otra forma. Se puede decir que la observación es el procedimiento empírico básico y co permite investigar el fenómeno en su manifestación externa.

Este método empírico ha estado presente durante todo el desarrollo de nuestra investigación: desde la observación del SCADA en sentido general y el módulo de AGDH en particular para determinar nuestro problema científico, hasta la interpretación de los resultados de nuestro trabajo.

Experimentación: El experimento es aquella clase de experiencia científica, en la cual se provoca deliberadamente algún cambio y se observa e interpreta su resultado con alguna finalidad cognoscitiva.

La experimentación resultó vital en nuestra investigación en el momento de validar la efectividad de la solución propuesta. Para ello, se prueban las tres estrategias con que cuenta el módulo de AGDH con una serie de datos de entrada, los cuales fueron utilizados posteriormente para examinar la utilidad de la nueva estrategia que se propone.

Entrevista: La entrevista es un típico método de interrogación. Los métodos de interrogación son aquellos que permiten obtener información, mediante preguntas directas a los sujetos incluidos en la muestra o especialistas en el tema motivo de cualquier investigación.

Una fuente importante de nuestro trabajo fueron las entrevistas realizadas a especialistas del SCADA *Guardián del Alba* así como a los creadores de la primara iteración del módulo de AGDH, Ing. Lazaro Abreu Reche, Ing. Abdelaziz de la Horra Díaz, Ing. Fernando Jiménez López, para tomar información acerca del módulo y sus particularidades.

Capítulo 1 Fundamentación Teórica

Introducción

En el presente capítulo quedan expuestos los principales conceptos manejados durante la investigación. Se caracteriza el módulo de AGDH del SCADA *Guardián del Alba* y su subsistema de balance de carga. También se hace un estudio de los patrones de software al servicio del balanceo. Además se hace un análisis de los algoritmos de toma de decisiones y tecnologías que permitirán el desarrollo de la solución.

1.1 SCADA *Guardián del Alba*.

En el presente, los continuos avances de la tecnología han complejizado los procesos industriales ampliando las posibilidades de automatización de estos. De aquí que se comenzaran a desarrollar complejos y útiles sistemas de control y adquisición de datos conocidos como SCADA.

El término SCADA proviene de las siglas en inglés de *"Supervisory Control And Data Acquisition"* (Control Supervisorio y Adquisición de Datos). Se trata de aplicaciones de *software* para el control de la producción, que se comunican con los dispositivos de campo y controlan los procesos desde la pantalla del ordenador. Además, estos sistemas son capaces de proporcionar información de la planta a diversos usuarios como: operadores, supervisores de control de calidad y administradores del sistema.

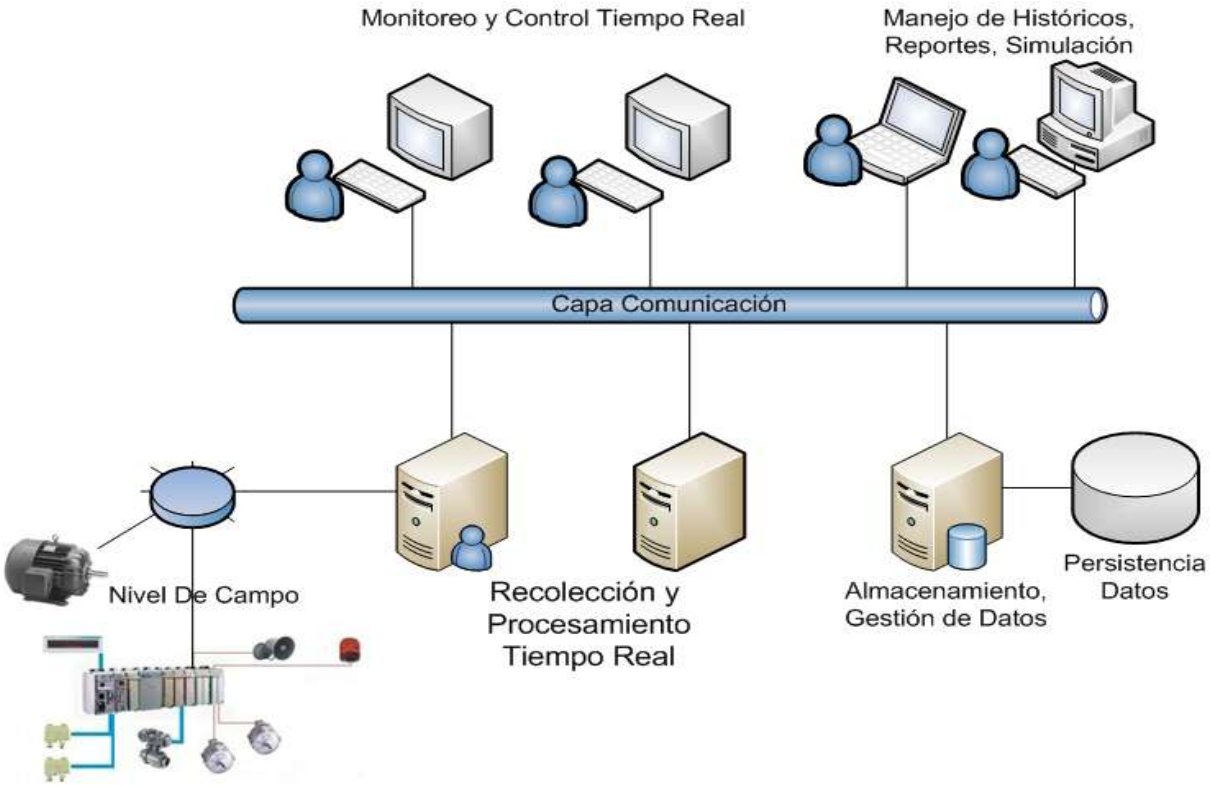


Figura 1. Representación de un SCADA.

En la figura anterior (Figura 1) aparece la representación de un SCADA a partir de los diferentes módulos que lo componen.

La UCI, desde su surgimiento, se ha convertido en un invaluable centro de desarrollo de software para nuestro país. Uno de los proyectos más importantes asumido por la Universidad es el *SCADA Guardián del Alba*, cuyo objetivo es lograr una aplicación informática con herramientas de software libre, para la supervisión de procesos industriales. Este proyecto está dividido en diferentes líneas de trabajo:

- Sabores de Linux
- Middleware
- Gráficos

- Drivers
- Reportes
- Base de Datos de Tiempo Real
- Base de Datos Históricos
- Comunicación OPC

Estas, a su vez, son las encargadas del desarrollo de su correspondiente módulo dentro del SCADA.

1.2 Módulo de Almacenamiento y Gestión de Datos Históricos.

Entre las líneas de trabajo mencionadas con anterioridad, la Base de Datos Históricos es la responsable del desarrollo del módulo del mismo nombre o módulo de AGDH. Este se encarga de procesar los datos para posteriormente almacenarlos en una base de datos como datos históricos. Además, almacena la información del sistema con el objetivo de que esta pueda ser empleada luego en la generación de reportes, tendencias, gestión de producción, etc.

El gráfico que se muestra a continuación (Figura 2) esquematiza el subsistema historiador, compuesto por un grupo de submódulos que permiten la interacción con los demás componentes del SCADA *Guardián del Alba*. [2]

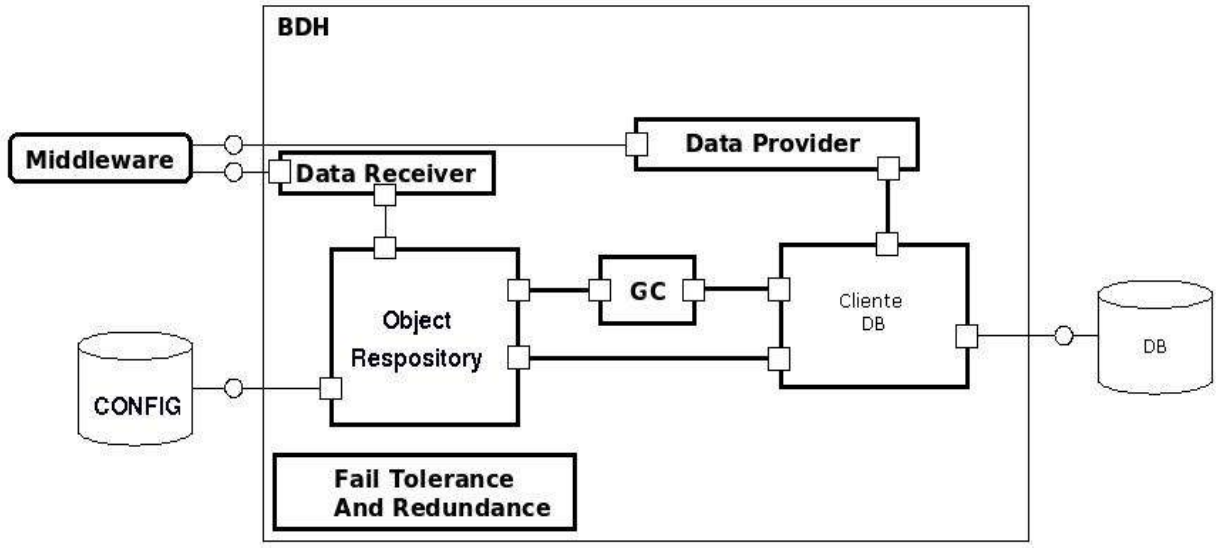


Figura 2. Subsistema de AGDH

1.2.1 Subsistema de balance de carga

El módulo de AGDH recibe los datos a través del Middleware. En ocasiones, se producen fallas en el proceso que dan origen a una avalancha de datos muy superior a la que este módulo puede asimilar. En tales casos se produce un retraso en el procesamiento y almacenamiento de los mismos, haciendo que se congestione el módulo y provocando la pérdida de estos datos. Con el fin de distribuir el trabajo en esos casos, se hizo necesario agregar una nueva funcionalidad al módulo.

Se le agregó entonces el balance de carga como submódulo adicional al AGDH. Este submódulo utiliza un balance centralizado donde cada nodo ejecuta el algoritmo o estrategia de balance y mantiene el estado global del sistema. Posee varias estrategias de balanceo: Random, Round Robin y Least Loaded, basadas en algoritmos aleatorios y cíclicos, permitiendo que una de ellas sea seleccionada en el momento de arrancar el sistema. La estrategia a utilizar se selecciona en la configuración inicial del módulo y puede ser cambiada en tiempo de ejecución.

1.3 Programación multi-hilo

Una hebra (*thread*) es un semi-proceso, que posee su propia pila (*stack*), y ejecuta un segmento de código. A diferencia de un proceso real, el *thread* normalmente comparte su memoria con otros *threads* (cuando se tienen varios procesos generalmente se tiene una porción de memoria diferente para cada uno de ellos). Un grupo de *threads* dentro de un mismo proceso comparten la misma memoria, por lo que pueden acceder a las variables globales. Todas estos *threads* se ejecutan en paralelo. [2]

La ventaja de usar un grupo de *threads* sobre el uso de un programa serial normal, es que algunas operaciones pueden ser ejecutadas en paralelo. De este modo, los eventos pueden ser manejados inmediatamente de su aparición. [2]

El uso de un grupo de *threads* en lugar de un grupo de procesos es favorable para el procesador, ya que el cambio entre *threads* es mucho más rápido que si se efectuara entre procesos. Además, la comunicación entre dos *threads* suele ser mucho más rápida y fácil que la comunicación entre dos procesos. [2]

Por otro lado, es importante destacar que por el hecho de que un grupo de *threads* comparten el mismo espacio de memoria, si una de ellas destruye el contenido de esa memoria el resto de los *threads* sufrirían el impacto. [2]

1.4 Balance de carga.

Una de las grandes preocupaciones en ambientes paralelos y distribuidos es el desarrollo de técnicas eficientes para distribuir procesos de un programa paralelo sobre múltiples procesadores. Estas técnicas se conocen como "*Balance de Carga"* y una definición precisa de ella la entregó M. Bozyigita [3]:

"Balance de carga es una técnica que acrecenta los recursos, explotando el paralelismo, y acortando el tiempo de respuesta mediante una distribución apropiada de la aplicación".

Es decir, el balance de carga se refiere a la técnica usada para compartir el trabajo a realizar entre varios procesos, ordenadores, discos u otros recursos, es el proceso de compartir los recursos computacionales (Figura 3). Con el advenimiento de las conexiones de alta velocidad de comunicación, se ha convertido en un beneficio para conectar equipos independientes de manera distribuida a través de un enlace de alta velocidad. Las principales ventajas de los sistemas distribuidos son: brindan un alto rendimiento, disponibilidad, y extensibilidad a bajo costo. Por lo tanto, la computación distribuida ha adquirido una

importancia creciente en la actualidad como modo preferido de la informática sobre la informática centralizada.

Un sistema de computación distribuida se compone de programas de software y datos dispersos en todos los equipos independientes y conectados a través de una red de comunicación. Algunos ordenadores pueden ser muy cargados, mientras que otros permanecen inactivos.

Mejorar el rendimiento es una de las cuestiones más importantes en los sistemas distribuidos. A menudo se puede elevar el rendimiento a un nivel aceptable, simplemente mediante la redistribución de la carga.

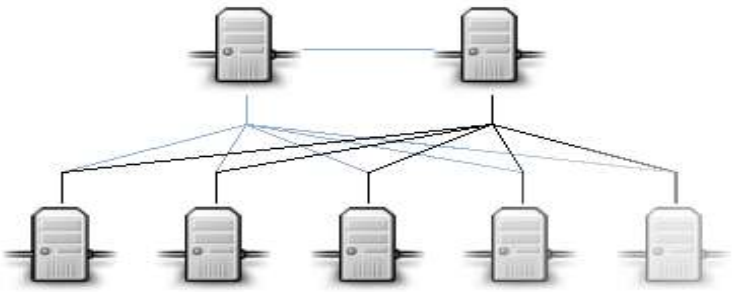


Figura 3. Balance de carga

1.4.1 Balance de carga estático y dinámico.

El balance de carga se puede clasificar en dos tipos: estático y dinámico. Se dice que se está en presencia de un balance de carga estático (Figura 4), cuando se tiene un conocimiento previo a la ejecución. Sin embargo, el balance de carga dinámico (Figura 5) en la reasignación de las tareas tiene en cuenta la carga de trabajo actual, se realiza sobre la marcha y se puede adaptar a los cambios que se presenten.

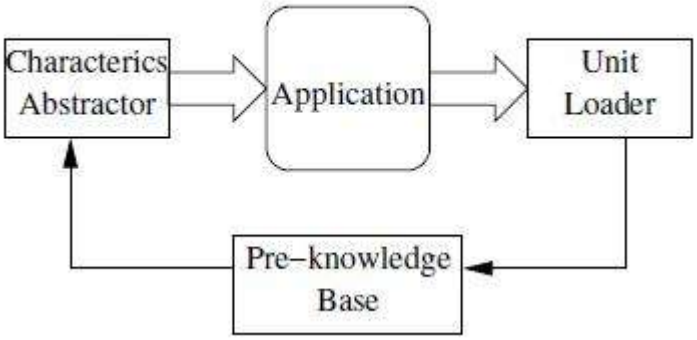


Figura 4. Balance de carga estático

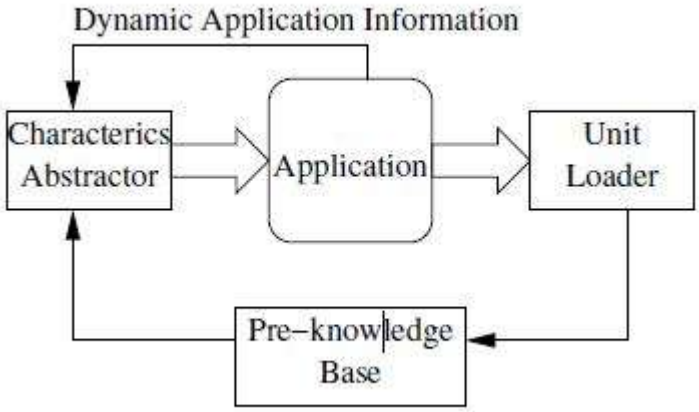


Figura 5. Balance de carga dinámico

1.5 Patrones de software en el servicio de balanceo.

El uso de patrones -por su naturaleza genérica de Problema-Solución-Reutilización- brinda para los sistemas distribuidos, desde su etapa de diseño, la capacidad de adaptación y posible extensión a su comportamiento.

Patrón Broker

Ayuda a resolver los problemas derivados del diseño de sistemas distribuidos y heterogéneos, referentes a la distribución de la infraestructura del software, simplificando las aplicaciones de sistemas distribuidos, y la comunicación fundamental soportados por productos y plataformas middleware.[4]

El servicio de balanceo diseña y representa el equilibrio que permite a los componentes acceder a los servicios que ofrecen otros componentes mediante invocaciones de servicio remotas y transparentes a la localización de los servidores, para cambiar, añadir o eliminar componentes en tiempo de ejecución. En el balanceo estos requerimientos son pasados a través de proxies al broker. A su vez, el broker busca para su localización al objeto servidor correspondiente y le pasa los requerimientos.

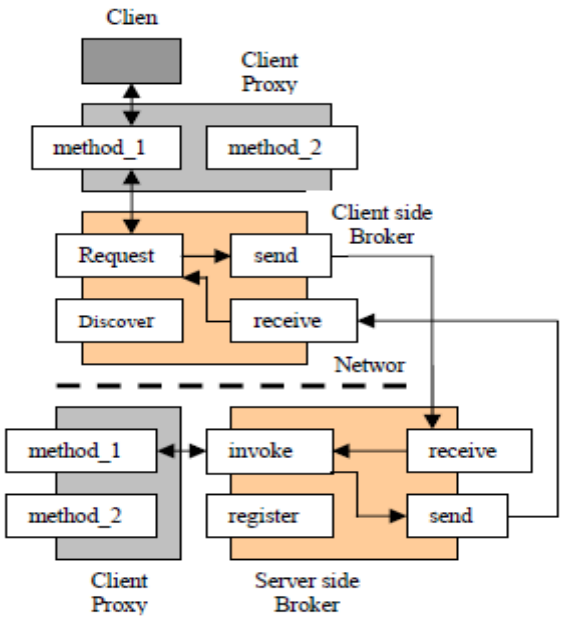


Figura 6. Patrón Broker

Patrón Interceptor

Permite agregar fácilmente funcionalidad al sistema para cambiar su comportamiento dinámicamente, sin necesidad de recopilarlo. Es decir, hace el cambio de comportamiento en tiempo de ejecución, ya sea interponiendo una nueva capa de procesamiento o cambiando el destino dinámicamente. Para ello, interpone objetos que pueden interceptar llamadas e insertar un procesamiento específico que puede estar basado en el análisis del contenido, además de redireccionar una llamada a un punto diferente. [4]

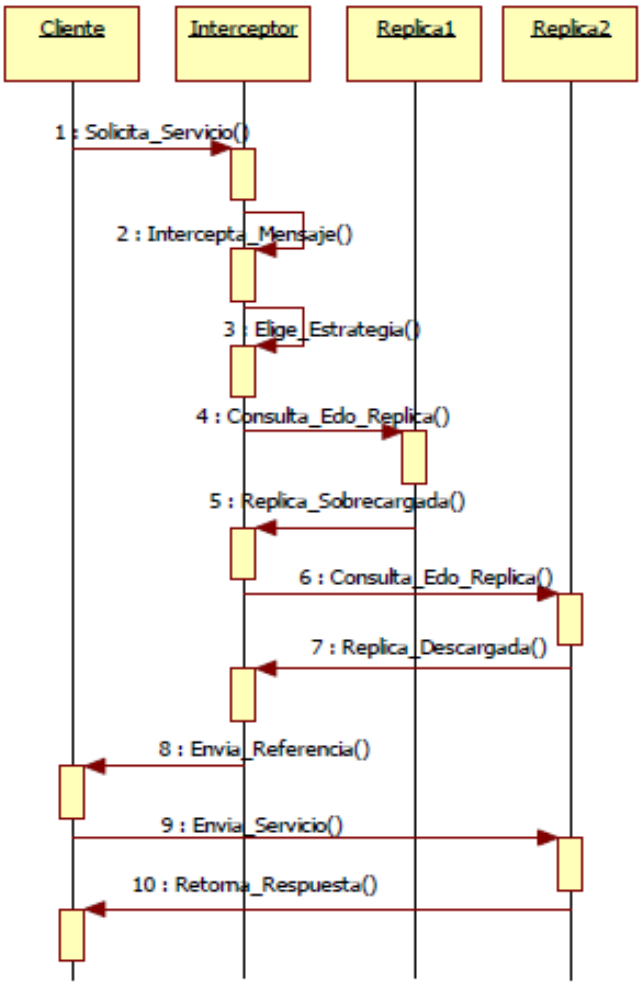


Figura 7. Diagrama de secuencia del patrón Interceptor

Patrón Estrategia

En general, no es común que todas las aplicaciones distribuidas exhiban las mismas condiciones de carga. Esto significa que algunas estrategias de balanceo son más aplicables a algunos tipos de aplicaciones que a otras.

Un balanceador de carga debe ser lo suficientemente flexible para soportar diferentes tipos de estrategias de balanceo, tanto estáticas como dinámicas. [4]

Las estrategias encapsuladas en este patrón son las siguientes:

Random

Esta estrategia es no adaptativa (no considera las condiciones de carga en tiempo de ejecución) y selecciona aleatoriamente una réplica de un grupo de réplicas para el envío de requerimientos. [4]

Round Robin

Estrategia no adaptativa que simplemente escoge una réplica de un grupo de réplicas para enviar los requerimientos del cliente, seleccionándola rotativamente de un grupo de objetos dado. [4]

Least Loaded (LL)

La estrategia LL es utilizada para el balanceo dinámico de carga. A diferencia de Random y Round Robin, la Least Loaded usa estrategia de balanceo de carga adaptativa. Como su nombre lo indica, esta estrategia elige dinámicamente al miembro de un grupo de objetos con la carga baja utilizando información en tiempo de ejecución. [4]

Estrategia basada en Fuzzy-Logic

Normalmente las estrategias son encapsuladas en el componente analizador de carga y este por su parte usa el patrón estrategia para configurar el algoritmo de balanceo que se usará cuando se tomen las decisiones de balancear. [4]

La propuesta primaria del analizador es determinar las condiciones de carga de las réplicas y grupos de implementación. Para ello, un soporte de múltiples estrategias de balanceo estará disponible. Dado que el conocimiento a priori de los requerimientos exhibidos por una aplicación distribuida no siempre está disponible, el patrón estrategia brinda la habilidad para dinámicamente agregar o remplazar las estrategias de balanceo soportadas por el analizador de carga dado. [4]

Este modelo de Fuzzy-Logic por lo general es utilizado para cálculos en aplicaciones que involucran cierta incertidumbre, siendo esta la razón por la que se propone como estrategia para el balanceo de carga incluyendo consigo la lógica difusa. Esto optimiza los recursos en sistemas a gran escala.

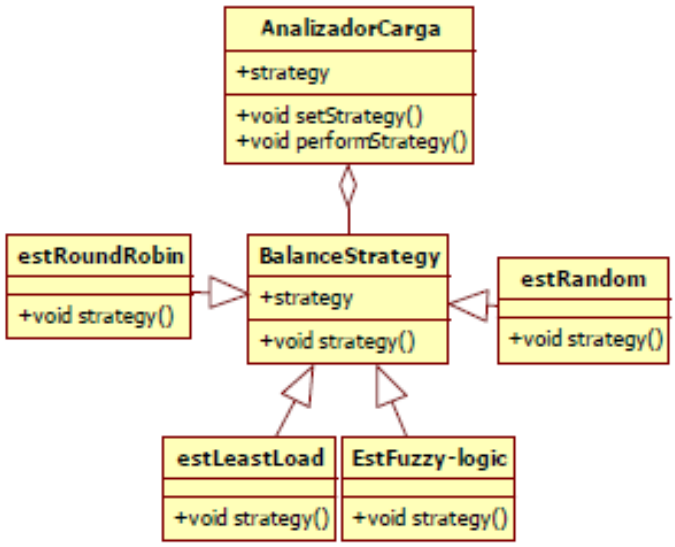


Figura 8. Diagrama de clases del patrón Estrategia

1.6 Inteligencia Artificial.

Se llama Inteligencia Artificial:

- Una de las áreas de las ciencias computacionales encargadas de la creación de hardware y software con comportamiento inteligentes.
- El estudio de la computación que permiten percibir, razonar y actuar
- Lo que estudia como lograr que las máquinas realicen tareas que, por el momento, son realizadas mejor por los seres humanos.

Si se busca el significado de inteligencia artificial en un diccionario, es probable que encuentre el siguiente planteamiento: "La capacidad de un ordenador u otro equipo para llevar a cabo las actividades que normalmente se piensa que requieren de inteligencia". Esta definición proviene del *American Heritage Dictionary* del idioma inglés, cuarta edición (Houghton Mifflin Company).

Desde el punto de vista de los objetivos, la IA puede considerarse como parte de la ingeniería o de la ciencia:

- El objetivo ingenieril de la IA es resolver problemas reales, actuando como un armamento de ideas acerca de cómo representar y utilizar el conocimiento, y de como ensamblar sistemas.

- El objetivo científico de la IA es explicar varios tipos de inteligencia. Determinar qué ideas acerca de la representación del conocimiento, del uso que se le da a éste, y del ensamble de sistemas explican distintas clases de inteligencia.

Algunas aplicaciones de la inteligencia artificial:

Tareas de la vida diaria:

- Percepción (visión y habla).
- Lenguaje natural (comprensión, generación, traducción).
- Sentido común.
- Control de un robot.

Tareas formales:

- Juegos (ajedrez, damas).
- Matemáticas (geometría, lógica, cálculo, demostración de propiedades).

Tareas de los expertos:

- Ingeniería (diseño, detección de fallos, planificación de manufacturación, control de procesos industriales).
- Análisis científico.
- Diagnosis médica.
- Análisis financiero.

Técnicas de resolución de problemas de la Inteligencia Artificial.

Uno de los resultados que surgieron de las primeras investigaciones en IA fue que la inteligencia necesita conocimiento. El conocimiento posee algunas propiedades poco deseables como:

- Es voluminoso.
- Es difícil caracterizarlo.
- Cambia.
- Se organiza de manera que se corresponde con la forma en que va a ser usado.

Una técnica de IA es un método que explota el conocimiento representado de manera que se cumpla que:

- Representa generalizaciones, es decir, no es necesario representar cada situación individual, sino que las situaciones que comparten propiedades importantes se agrupan.
- Debe ser entendido por las personas que lo provean.
- Puede ser modificado para corregir errores y reflejar cambios en el mundo.

- Puede usarse en muchas situaciones aún sin ser totalmente exacto o completo.
- Puede usarse para superar su propio volumen, y disminuir el rango de posibilidades que normalmente deben considerarse.

Se pueden caracterizar las técnicas de IA con independencia del problema a tratar. Para solucionar problemas complicados, los programas que utilizan las técnicas de IA presentan numerosas ventajas con respecto a los que no lo hacen:

- Son menos frágiles, es decir, que no se despistan frente a una perturbación pequeña de la entrada.
- El conocimiento del programa es comprendido fácilmente por la gente.
- Usa generalizaciones.
- Tiene facilidad de extensión.

Entre las ramas de la Inteligencia Artificial se encuentran los algoritmos de toma de decisiones. Que son ampliamente utilizados para el control de complejos procesos industriales, búsqueda, optimización y para solución de conflictos.

1.7 Toma de decisiones

Tanto técnicamente como en nuestra vida diaria, hay que enfrentar decisiones, ya sean grandes o pequeños los problemas que haya que solucionar. En cualquier caso, para tomar decisiones es necesario ante todo realizar un análisis de la situación en cuestión, pues una vez que se lleve a cabo el análisis se podrá identificar y formular el problema con mayor claridad.

Ya definido este, el análisis de sus aspectos relevantes permitirá evaluar las posibles soluciones e identificar cuál se ajusta mejor al resultado esperado para finalmente formularla y aplicarla.

Realizar una buena toma de decisiones permite obtener mejores soluciones y otorga algo de control sobre los problemas. Un componente fundamental de la toma de decisiones es tener en cuenta los efectos futuros de estas, ya que podrían estar relacionadas con posteriores decisiones en otros niveles del problema, es decir, pudieran conllevar a otro problema en un determinado momento.

1.7.1 Algoritmos de toma de decisiones

La toma de decisiones no es un proceso exclusivo de los humanos. Algunas áreas del conocimiento, como la IA, desarrollan algoritmos capaces reproducir este proceso.

Algunos de estos algoritmos son:

1. Árboles de decisión.
2. Redes Bayesianas.
3. Sistemas de Markov.
4. Sistemas basados en reglas.
5. Máquinas de estado finito.
6. Lógica difusa.

Árboles de decisión

Técnicamente, los árboles de decisión permiten analizar decisiones secuenciales basadas en el uso de resultados y probabilidades asociadas. El uso de los mismos trae consigo una serie de ventajas:

- Resumen los ejemplos de partida, permitiendo la clasificación de nuevos casos siempre y cuando no existan modificaciones sustanciales en las condiciones bajo las cuales se generaron los ejemplos que sirvieron para su construcción.
- Facilitan la interpretación de la decisión adoptada.
- Proporcionan un alto grado de comprensión del conocimiento utilizado en la toma de decisiones.
- Explican el comportamiento respecto a una determinada tarea de decisión.
- Reducen el número de variables independientes.

Los árboles de decisión se utilizan en cualquier proceso que implique toma de decisiones, ejemplos de estos procesos son:

- Búsqueda binaria.
- Sistemas expertos.
- Árboles de juegos.

Los árboles de decisión son generalmente binarios, es decir, cuentan con dos opciones. Sin embargo, esto no significa que no puedan existir árboles de tres o más opciones.

Redes Bayesianas

Las Redes Bayesianas (RB) modelan un fenómeno mediante un conjunto de variables y las relaciones de dependencia entre ellas. Dado este modelo, se puede hacer inferencia bayesiana, es decir, estimar la probabilidad posterior de las variables no conocidas, en base a las variables conocidas. Estos modelos pueden tener diversas aplicaciones, para clasificación, predicción, diagnostico, etc. Además, pueden proporcionar información interesante en cuanto a cómo se relacionan las variables del dominio, que pueden ser interpretadas en ocasiones como relación de causa-efecto.

Existen distintos tipos de RB:

- Naive Bayes = bayes “ingenuo” o Idiot's Bayes

  Forma de “**V**” => 2^n estados en el nodo inferior

- DBNs = Redes Bayesianas Dinámicas

  Cambian con el tiempo (t, t+1, t+2...)

  Lo pasado en t, tiene relación con lo que suceda en t+1

- Redes Gaussianas = distribución gaussiana

  Para nodos con variables continuas

- Cadenas de Markov = subconjunto de las RB

Además, tienen un amplio uso como por ejemplo en la meteorología, en los sistemas de aires acondicionados, etc.

Las RB son una representación gráfica de dependencias para razonamiento probabilístico, en la cual los nodos representan variables aleatorias y los arcos representan relaciones de dependencia directa entre las variables (Figura 9).

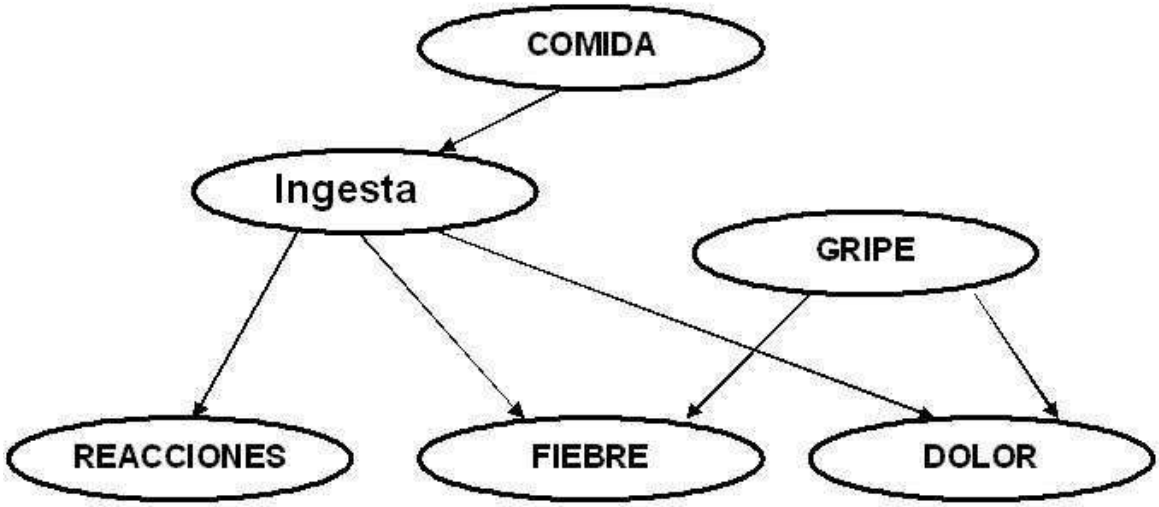


Figura 9. Ejemplo de una RB

Sistemas de Markov

Un proceso de Markov es un proceso que se va moviendo de estado en estado dependiendo exclusivamente de los N estados anteriores.

La diferencia de un proceso de orden 1 con un proceso determinista es que la elección del estado final se hace de forma probabilística, no determinista. En un proceso de Markov, aparte de los estados se tiene en cuenta la matriz de transiciones entre estados, que indica la probabilidad de pasar de un estado a otro o a sí mismo. Además, es necesaria una matriz que indique cómo se encuentra el sistema al inicio.

Modelos Ocultos de Markov (HMM)

Este tipo de sistema trabaja considerando que la producción interna de fonemas es una secuencia de estados ocultos (no observables) y que los sonidos resultantes son una secuencia observable generada

por los estados ocultos. El número de estados ocultos puede ser diferente del número de estados observables.

Los modelos ocultos de Markov (HMM) constituyen una herramienta de modelización altamente flexible, inicialmente utilizada en el campo del reconocimiento automático del habla, que ha encontrado en los últimos años numerosas aplicaciones en áreas científico-técnicas muy diversas, aunque su utilización en ecología es aún escasa. Varios son los usos y aplicaciones de los HMM, en la era moderna; como es el caso del reconocimiento automático de la voz anteriormente mencionado, reconocimiento de gestos, etc.

Sistemas basados en reglas

En nuestra vida diaria existen muchas situaciones complejas gobernadas por reglas deterministas: sistemas de control de tráfico, sistemas de seguridad, transacciones bancarias, etc. Los sistemas basados en reglas son una herramienta eficiente para tratar estos problemas. Las reglas deterministas constituyen la más sencilla de las metodologías utilizadas en sistemas expertos. La base de conocimiento contiene las variables y el conjunto de reglas que definen el problema, y el motor de inferencia obtiene las conclusiones aplicando la lógica clásica a estas reglas.

Se entiende por regla una proposición lógica que relaciona dos o más objetos e incluye dos partes: la premisa y la conclusión. Cada una de estas partes consiste en una expresión lógica con una o más afirmaciones objeto-valor conectadas mediante los operadores lógicos y, o, o no. Una regla se escribe normalmente como "Si…, entonces…".

Los sistemas basados en reglas tienen una serie de aplicaciones diversas, pero fundamentalmente son usados para:

1. Son usados en el Modelado de sistemas para representar escenarios de actuación con variables complejas, en los cuáles no basta con usar realidades absolutas como lo sería un cierto o falso. También se utilizan en aplicaciones de control, ya que suelen estar basadas en el punto descrito anteriormente.

2. Pueden ser usados para la categorización de elementos, detectar patrones etc.

3. Como agentes de usuario, los cuáles van a proporcionar una serie de ayudas a los expertos en una determinada materia para que tomen las decisiones que ellos consideren oportunas. En este caso, estos

sistemas van a ofrecer una ayuda, pero la decisión final la va a tomar el experto en cuestión. Como ejemplo cabe destacar el ámbito financiero.

Se puede afirmar que la principal ventaja de estos sistemas es que son muy simples, mientras que sus principales problemas están en que es muy limitado, difícil de depurar y modificar, es no adaptativo y su escalabilidad es nula.

Máquinas de Estado Finito

Las Máquinas de Estado Finito, conocidas como Finite State Machines (FSM) por su traducción al inglés, sirven para realizar procesos bien definidos en un tiempo discreto. Reciben una entrada, hacen un proceso y entregan una salida.

Una Máquina de Estado Finito en un modelo abstracto para la manipulación de símbolos, permite saber si una cadena pertenece a un lenguaje o pueden generar otro conjunto de símbolos como resultado. Una Máquina de Estado Finito también puede ser llamada como Autómata Finito, se puede utilizar ambos términos de forma indistinta.

Los autómatas se caracterizan por tener un estado inicial. Reciben una cadena de símbolos, cambian de estado por cada elemento leído o pueden permanecer en el mismo estado. También tienen un conjunto de estados finales o aceptables que indican si una cadena pertenece al lenguaje al final de una lectura.

Una Máquina de Estado Finito sólo puede estar en un estado en cada instante y está compuesta por 4 elementos principales:

- Estados que definen el comportamiento y pueden producir acciones.
- Transiciones de estado que son movimientos de un estado a otro.
- Reglas o condiciones que deben cumplirse para permitir un cambio de estado.
- Eventos de entrada que son externos o generados internamente, que permiten el lanzamiento de las reglas y consienten las transiciones.

Brinda una serie de ventajas como son:

- Su simplicidad la hace fácil para los desarrolladores sin experiencia.

- Predictibilidad (en FSM deterministas).
- Las FSM son rápidas de diseñar e implementar.
- Las FSM son relativamente flexibles en su implementación.
- Bajo uso del procesador.
- Es fácil determinar si se puede llegar o no a un estado.

Su uso trae consigo algunas desventajas:

- La naturaleza predecible de las FSM deterministas puede no resultar conveniente en algunos dominios.
- Si se implementa un sistema grande usando FSM puede ser difícil de administrar y mantener sin un buen diseño.
- No es apropiada para todos los dominios de problema, sólo debe ser usada cuando el comportamiento de un sistema puede ser descompuesto en estados separados con condiciones bien definidas para las transiciones.
- Las condiciones para las transiciones entre estados son rígidas, significando que están fijadas.

Lógica difusa

La lógica difusa o lógica borrosa, es una rama de la Inteligencia Artificial y es un subconjunto de la lógica convencional, que ha sido extendido para manejar el concepto de verdad parcial. La lógica difusa permite gobernar un sistema por medio de reglas de sentido común que se refieren a cantidades indefinidas.

Las reglas involucradas en un sistema difuso pueden ser aprendidas por sistemas adaptativos que aprenden al observar cómo operan las personas los dispositivos reales, aunque estas reglas pueden también ser formuladas por un experto humano. La lógica difusa es entonces definida como un sistema matemático que modela funciones no lineales, que convierte unas entradas en salidas acorde a los planteamientos lógicos que usan el razonamiento aproximado.

La lógica difusa se utiliza cuando la complejidad del proceso en cuestión es muy alta y no existen modelos matemáticos precisos, para procesos altamente no lineales y cuando se envuelven definiciones y conocimiento no estrictamente definido (impreciso o subjetivo).

En cambio, no es una buena idea usarla cuando algún modelo matemático ya soluciona eficientemente el problema, cuando los problemas son lineales o cuando no tienen solución.

Esta técnica se ha empleado con bastante éxito en la industria, principalmente en Japón, y cada vez se usa más efectivamente en gran multitud de campos. La primera vez que se le dio un uso importante fue en el metro japonés, obteniéndose excelentes resultados. A continuación algunos ejemplos de su aplicación:

- Sistemas de control de acondicionadores de aire.
- Sistemas de foco automático en cámaras fotográficas.
- Electrodomésticos familiares (frigoríficos, lavadoras...).
- Optimización de sistemas de control industriales.
- Sistemas de reconocimiento de escritura.
- Mejora en la eficiencia del uso de combustible en motores.
- Sistemas expertos del conocimiento (para simular el comportamiento de un experto humano).
- Tecnología informática.
- Bases de datos difusas (para almacenar y consultar información imprecisa).
- En general, en la gran mayoría de los sistemas de control que no dependen de un Sí/No.

Conjuntos Difusos

Un conjunto difuso no es más que un conjunto que puede contener elementos con grados parciales de pertenencia, a diferencia de los conjuntos clásicos en los que los elementos pueden “pertenecer” o “no pertenecer” a dichos conjuntos. [5]

Los conjuntos difusos presentan una función de membresía o pertenencia que determina el grado en que un elemento es miembro o pertenece al conjunto, siendo esta función, una curva que determina el grado de pertenencia de los elementos de un conjunto. Se denota generalmente por μ y puede adoptar valores entre 0 y 1.

Por ejemplo, para un conjunto clásico se tendría lo siguiente: [5]

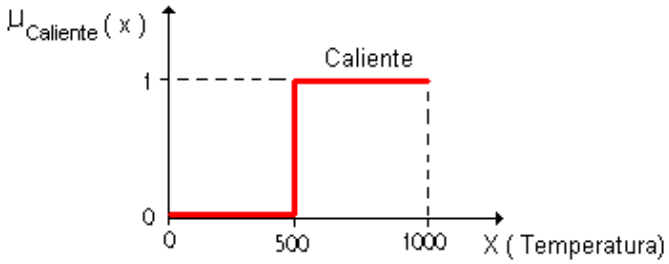


Figura 10. Gráfica de función de pertenencia para un conjunto clásico.

El conjunto se definiría así: **Caliente = { x | x > 500 }** , Temp [°C]

$\mu_{Caliente}$(x) = 1, x ∈ Caliente **$\mu_{Caliente}$(x) = 0, x ∉ Caliente**

$\mu_{Caliente}$(x) = { 0 , 1} ➔ Solo toma estos valores.

Figura 11. Definición de la Figura 10.

Para un Conjunto Difuso se tendría lo siguiente: [5]

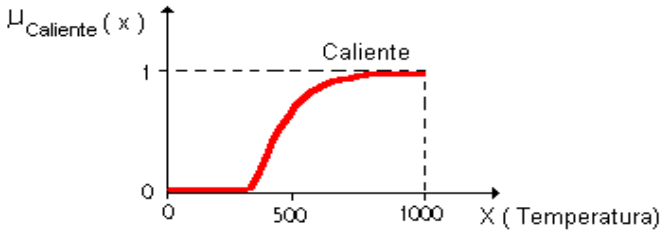


Figura 12. Gráfica de función de pertenencia para un conjunto difuso.

El conjunto se definiría así: **Caliente = { (x, $\mu_{Caliente}$(x)) | x ∈ U }**

U = [0 , 1000] ➔ Universo de Discurso.

$\mu_{Caliente}$(x) = [0 , 1] ➔ Rango de Valores. Temp [°C]

Figura 13. Definición de la Figura 12.

Donde el universo de discurso es el Conjunto de valores que puede tomar una variable.

1.7.2 Herramientas para el trabajo con la Lógica Difusa.

Para el desarrollo de sistemas basados en lógica difusa es muy útil el uso de bibliotecas, lenguajes y herramientas que ayuden en el diseño e implementación. Existen muchas herramientas tanto comerciales como gratuitas. Si se desea facilitar la programación se cuenta con una serie de bibliotecas que simplifican esta tarea. A continuación se describen algunas de estas herramientas y bibliotecas.

Free Fuzzy Logic Library (FFLL)

Free Fuzzy Logic Library (FFLL) es una biblioteca de lógica difusa de código abierto, escrita en C++. Está optimizada para aplicaciones en las que la velocidad es crítica, como vídeo juegos. [6]

FFLL fue diseñada para ser rápida, por lo tanto, las tablas de búsqueda se utilizan siempre que sea posible, sacrificando parte de la memoria por el bien de la velocidad. Almacena las reglas en una matriz unidimensional. Esto evita las complejidades de la dinámica de la asignación de una matriz multidimensional en C / C + + cuando el número de dimensiones no se conoce de antemano. Para eliminar cálculos adicionales, FFLL no calcula el valor de salida defuzzified hasta que se solicita. Por ejemplo, si un modelo tiene cuatro variables de entrada y el valor de salida se calcula cada vez que cambia un valor de entrada, se estarían realizando tres cálculos innecesarios.

Dado que todas las clases de FFLL contienen información (por ejemplo, las reglas) que se comparte entre los hijos, cada hijo mantiene sus valores de entrada propia y cada hijo con seguridad puede estar en un subproceso independiente. [6]

FFLL puede cargar archivos de la Lengua Control Difuso (FCL, Fuzzy Control Language) tal como se define en la norma International Electrotechnical Commission (IEC) 61131-7 Estándar Internacional para la Programación del Control Fuzzy. [6]

JFuzzyQt

Esta es una biblioteca de código abierto. Es un clon de FFLL y al igual que esta carga ficheros compatibles con el estándar FCL. Esta biblioteca contiene una serie de funciones que facilitan su trabajo, y al ser un clon de FFLL gana la ventaja de tener sus funcionalidades además de incluir la corrección de algunos de los errores existentes en esta biblioteca.

XFuzzy

El entorno Xfuzzy es un desarrollo del centro Nacional de Microelectrónica del CSIC (Centro Superior de Investigaciones Científicas) de España. Xfuzzy es un entorno de desarrollo para sistemas de inferencia basados en lógica difusa. Está formado por un conjunto de herramientas que facilitan las distintas etapas del proceso de diseño, desde su descripción inicial hasta la implementación final. Actualmente se encuentra disponible la versión Xfuzzy 3.0, desarrollada en Java.

Xfuzzy propone una metodología de desarrollo de sistemas borrosos basada en cuatro etapas: descripción, verificación, ajuste y síntesis. La etapa de descripción ofrece herramientas visuales para la definición del sistema borroso. La etapa de verificación permite la simulación del sistema. La etapa de ajuste permite la utilización de algoritmos de aprendizaje para configurar las variables. Por último, la etapa de síntesis incluye herramientas para generar descripciones en lenguaje de alto nivel para implementaciones de software y hardware. Xfuzzy está basado en el lenguaje XFL 3, que permite definir bases de reglas jerárquicas, modificadores lingüísticos, funciones de pertenencia y métodos de defuzzificación. Una de las características más estable es el amplio abanico de técnicas y métodos disponibles en este lenguaje. [7]

JFuzzyLogic

Es un paquete de lógica difusa escrito en Java que implementa la especificación IEC 1131p7 de lenguaje de control difuso FCL. Utilizar el lenguaje FCL libera al usuario de tener que conocer las clases de implementación en Java con las que codificar las reglas difusas y puede limitarse a escribir un sencillo

archivo de texto donde definirlas utilizando un lenguaje de reglas if…then. Permite la selección de distintas funciones de pertenencia, métodos de defuzificación y métodos de implicación.

FuzzyTECH

FuzzyTECH es un producto propiedad de INFORM GmbH and Inform Software Corp. especializado en el desarrollo de aplicaciones técnicas que utilizan lógica difusa. Esta herramienta emplea una representación de números borrosos a través de funciones de pertenencia trapezoidales. Una de las características, a la FuzzyTECH presta especial atención es a la fuzzificación. La herramienta ofrece diferentes técnicas para el cálculo eficiente de la fuzzificación, e incluye algunas orientadas a la implementación hardware. FuzzyTECH permite usar muchos de los métodos de defuzzificación más habituales (CA, COA y MOM), así como otros utilizados para el reconocimiento de patrones (HyperCoM). [7]

Una de las características fundamentales de FuzzyTECH es la generación automática de código C. Cuando este se compila se obtiene resultados muy eficientes, lo cual es una característica muy atractiva para el desarrollo de aplicaciones profesionales. Además de versiones simplificadas con objetivos académicos, FuzzyTECH también ofrece extensiones para sistemas de control embebidos y desarrollados para la industria de la automoción. [7]

Fuzzy Control Language (FCL)

Fuzzy Control Language, es un estándar para la programación de control difuso, publicado por la (IEC). Las especificaciones de la sintaxis FCL puede encontrarse en el documento IEC 61131-7. [8]

1.8 Metodologías y herramientas a utilizar

Para la realización de una aplicación o cualquier sistema informático se deben definir las herramientas y las metodologías que serán de mayor utilidad para su desarrollo.

1.8.1 Sistema operativo

GNU/Linux

Hasta hace poco tiempo, GNU/Linux era utilizado básicamente por desarrolladores de software. En la actualidad los usuarios de este sistema operativo han ido más allá del término de trabajo o empresarial.

Empieza a ser usado por millones de personas atraídas por la singularidad de un sistema que permite un uso activo y consciente de las herramientas informáticas, en lugar de seguir sometidos al consumo pasivo y frustrante de sistemas rígidos, hartas de pantallazos azules de error, de los virus, del software caro y poco funcional, de supuestas novedades que se limitan a corregir algunos errores de la versión anterior (y a añadir otros nuevos), hartas de ser consideradas piratas por compartir programas con tus vecinos o con tus amigos.[9]

Características de Linux

- **Multitarea:** Linux utiliza la llamada multitarea preventiva, la cual asegura que todos los programas que se están utilizando en un momento dado serán ejecutados, siendo el sistema operativo el encargado de ceder tiempo de microprocesador a cada programa.

- **Multiplataforma:** Las plataformas en las que en un principio se puede utilizar Linux son 386-, 486-. Pentium, Pentium Pro, Pentium II, Amiga y Atari. También existen versiones para su utilización en otras plataformas, como amd64, Alpha, ARM, MIPS, PowerPC y SPARC.

- **Política de copia en escritura para la compartición de páginas entre ejecutables:** esto significa que varios procesos pueden usar la misma zona de memoria para ejecutarse. Cuando alguno intenta escribir en esa memoria, la página se copia a otro lugar. Esta política de copia en escritura tiene dos beneficios: aumenta la velocidad y reduce el uso de memoria.

- **Todo el código fuente está disponible**, incluyendo el núcleo completo y todos los drivers, las herramientas de desarrollo y todos los programas de usuario; además todo ello se puede distribuir libremente. Hay algunos programas comerciales que están siendo ofrecidos para Linux actualmente sin código fuente, pero todo lo que ha sido gratuito sigue siendo gratuito.

- **Consolas virtuales múltiples:** varias sesiones de login a través de la consola entre las que se puede cambiar con las combinaciones adecuadas de teclas (totalmente independiente del hardware de video). Se crean dinámicamente y se pueden tener hasta 64.

- **Multiusuario:** muchos usuarios usando la misma máquina al mismo tiempo.

- **Multiprocesador:** soporte para sistemas con más de un procesador está disponible para Intel, AMD y SPARC.

- **Protección de la memoria entre procesos**, de manera que uno de ellos no pueda colgar el sistema.

- **Carga de ejecutables por demanda:** Linux sólo lee del disco aquellas partes de un programa que están siendo usadas.

- **Librerías compartidas de carga dinámica** (DLL's) y librerías estáticas.

- **Sistema de archivos de CD-ROM** que lee todos los formatos estándar de CD-ROM.

Este sistema tiene una serie de ventajas sobre el sistema operativo Windows, entre ellas se encuentran:

Seguridad

- Los virus creados hasta el momento están principalmente dirigidos a servidores que poseen sistema operativo Windows, por lo que el usuario puede prescindir por completo de programas de antivirus.

- La plataforma Linux es más robusta, lo que hace más difícil que algún intruso pueda violar el sistema de seguridad.

- Las actualizaciones constantes del sistema permiten que el software esté permanentemente protegido.

Rapidez

- La eficiencia de su código fuente hace que la velocidad de las aplicaciones Linux sean superiores a las que corren sobre Windows.

Economía

- Requieren menor mantenimiento. En servidores Windows es más costoso debido a que es necesaria una frecuente atención y monitoreo contra ataques de virus, hackers y errores de código, instalación y actualización de parches y service packs.

- El software Linux, así como un sin número de aplicaciones son de código abierto.

Estabilidad

- Esta es una de sus prioridades, por lo que se realizan actualizaciones fáciles de implementar, que permiten que los usuarios trabajen de manera confiada y segura. Esto posibilita que puedan prescindir de acciones tales como reiniciar o formatear la PC, tan comunes en Windows. Esto se debe a que GNU/Linux realmente funciona sin inconvenientes luego de varios años de haber sido instalado, y recibiendo las últimas novedades que modifican aplicaciones, entornos gráficos y demás, manteniendo actualizado el sistema.

1.8.2 Metodología de desarrollo.

En el ciclo de vida del software se deben completar una serie de tareas para obtener un producto. A menudo, se dice que los distintos componentes de software deben pasar por distintas fases o etapas durante el ciclo de vida. Cada una de esas tareas puede ser abordada y resuelta de múltiples maneras, con distintas herramientas y utilizando distintas técnicas. Es necesario saber cuándo se puede dar por concluida una tarea, quién debe realizarla, qué tareas preceden a una dada, qué documentación se utilizaría para llevar a cabo esa tarea. Se está hablando de detalles organizativos, de un estilo de hacer las cosas. Yendo un poco más allá de un simple estilo, formalizando ese estilo añadiendo algo de rigurosidad y normas se obtiene una metodología.

Metodologías modernas que se pueden destacar:

- Desarrollo Rápido de Aplicaciones (Rapid Application Development - RAD).
- Scrum
- Programación extrema. (Extreme Programming - XP)
- Proceso Racional Unificado. (Rational Unified Process - RUP)
- Proceso Ágil Unificado. (Agile Unified Process - AUP)

Desarrollo Rápido de Aplicaciones (Rapid Application Development - RAD).

El **Desarrollo Rápido de Aplicaciones** (RAD) es una metodología de desarrollo de software, que implica el desarrollo iterativo y la construcción de prototipos. El RAD es un término originalmente utilizado para describir un proceso de desarrollo de software introducido por James Martin en 1991.

Principios básicos:

- Su objetivo clave es un rápido desarrollo y entrega de una alta calidad en un sistema de relativamente bajo coste de inversión.
- Intenta reducir el riesgo inherente del proyecto partiéndolo en segmentos más pequeños y proporcionar más facilidad de cambio durante el proceso de desarrollo.
- Orientación dedicada a producir sistemas de alta calidad con rapidez, principalmente mediante el uso de iteración por prototipos (en cualquier etapa de desarrollo), promueve la participación de los usuarios y el uso de herramientas de desarrollo computarizadas.
- Hace especial hincapié en el cumplimiento de la necesidad comercial, mientras que la ingeniería tecnológica o la excelencia es de menor importancia.
- El control de proyecto implica el desarrollo de prioridades y la definición de los plazos de entrega. Si el proyecto empieza a aplazarse, se hace hincapié en la reducción de requisitos para el ajuste, no en el aumento de la fecha límite.
- La participación activa de los usuarios es imprescindible.
- Realiza iterativamente la producción de software, en lugar de colgarse de un prototipo.
- Produce la documentación necesaria para facilitar el futuro desarrollo y mantenimiento.

Rational Unified Process (RUP)

RUP mantiene al equipo enfocado en producir incrementalmente software operativo a tiempo, con las características requeridas y con la calidad requerida. Las mejores prácticas probadas en la industria, contenidas en RUP, incorporan las lecciones aprendidas de cientos de líderes de la industria y miles de proyectos. Ya no hay necesidad de re-inventar soluciones a desafíos de la ingeniería de software bien conocidos. Siguiendo el acercamiento al desarrollo iterativo de RUP, es posible entregar a tiempo y con confianza el software.

Características y Beneficios

No existen dos proyectos de desarrollo de software que sean iguales. Cada uno tiene prioridades, requerimientos y tecnologías muy diferentes. Sin embargo, en todos los proyectos se debe minimizar el

riesgo, garantizar la predictibilidad de los resultados y entregar un software de calidad superior a tiempo. RUP es una plataforma flexible de procesos de desarrollo de software que ayuda proveyendo guías consistentes y personalizadas de procesos para todo el equipo de proyecto.

- **Las prácticas más probadas de la industria**. Son las mejores prácticas de desarrollo adoptadas en cientos de proyectos mundialmente. La metodología RUP se convirtió rápidamente en el estándar de facto para el proceso de desarrollo en la industria de software.

- **Proceso en práctica**. Diferente de otras metodologías comerciales, la plataforma RUP hace que el proceso sea práctico con bases de conocimiento y guías para ayudar en el despegue de la planificación del proyecto, integrar rápidamente a los miembros del equipo y poner en acción el proceso personalizado.

- **Se adapta a las necesidades de los proyectos**. Solo la plataforma RUP proporciona un framework de proceso configurable que permite seleccionar e implantar los componentes específicos de proceso necesarios para proporcionar un proceso consistente y customizado para cada equipo y proyecto.

Una de las mejores prácticas centrales de RUP es la noción de desarrollar iterativamente. RUP organiza los proyectos en términos de disciplinas y fases, consistiendo cada una en una o más iteraciones. Con esta aproximación iterativa, el énfasis de cada workflow variará a través del ciclo de vida. La aproximación iterativa ayuda a mitigar los riesgos en forma temprana y continua, con un progreso demostrable y frecuentes *releases* ejecutables.

Sistemas Operativos y Plataformas de Hardware apropiadas:

- HP-UX
- Linux
- Sun Solaris
- Windows 2000
- Windows NT
- Windows XP

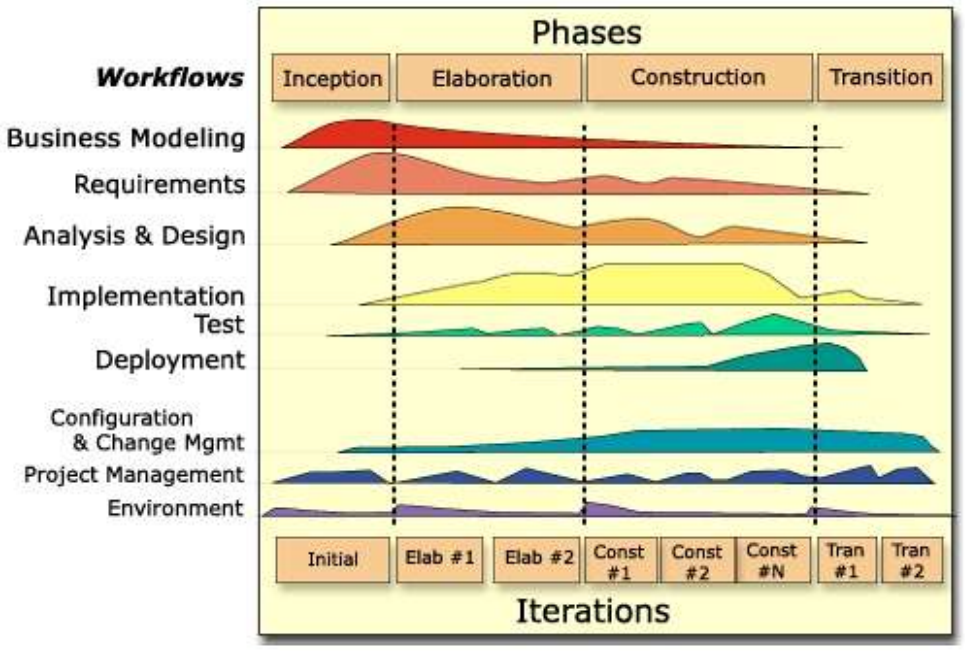


Figura 14. Rational Unified Process (RUP)

1.8.3 Lenguaje de modelado

UML

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad.

UML es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema software. [10]

UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software, tal como el Proceso Unificado Racional o RUP que lo utiliza como lenguaje de modelado, pero no especifica en sí mismo qué metodología o proceso usar.

Figura 15. Lenguaje Unificado de Modelado (UML)

1.8.4 Herramientas CASE

Las herramientas CASE (Computer Aided Software Engineering) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software, reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. [11]

Visual Paradigm.

Es una herramienta CASE para visualizar y diseñar elementos de software, para ello hace uso del lenguaje de modelado UML. Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Facilita la interoperabilidad con otras herramientas CASE como el Rational Rose y puede ser integrada con herramientas Java. Posee además gran facilidad de uso, acelera el desarrollo de aplicaciones, ya que sirve de puente visual entre arquitectos, analistas y diseñadores de sistemas de información, haciendo el trabajo más fácil y dinámico.

Visual Paradigm es una herramienta fácil de usar que soporta la última notación UML 2.1, ingeniería inversa, generación de código, importación desde Rational Rose, exportación/importación XMI, generador

de informes, editor de figuras, integración con MS Visio, plug-in, integración IDE con Visual Studio, IntelliJ IDEA, Eclipse, NetBeans y otros.

Características

- Producto de calidad.
- Soporta aplicaciones web.
- Las imágenes y reportes generados son de muy buena calidad.
- Varios idiomas.
- Generación de código para Java y exportación como HTML.
- Fácil de instalar y actualizar.

Rational Rose

Es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Cubre todo el ciclo de vida de un proyecto:

- Concepción y formalización del modelo.
- Construcción de los componentes.
- Transición a los usuarios
- Certificación de las distintas fases.

1.8.5 Autotools para codificación y compilación.

Herramientas como *automake* y *autoconf* se encuentran disponibles en muchos de los proyectos *open sources*. La ventaja en el uso de estas herramientas se debe a que ayudan a la portabilidad de las aplicaciones a nivel de código fuente, abstrayéndose en la medida de lo posible, de las versiones de las herramientas tradicionales disponibles en cada sistema operativo tipo Unix. Las mismas le facilitan al desarrollador su trabajo para dejar disponible un *script* de autoconfiguración de su aplicación. [12]

aclocal

aclocal es una utilidad que permite generar automáticamente el archivo aclocal.m4. Busca macros en todos los archivos .m4 del directorio en donde se ejecuta y finalmente busca en el archivo configure.in. Todas esas macros se incluirán en el archivo aclocal.m4, así como todas las macros de las cuales dependen. De esta forma, se puede emplear el mismo archivo configure.in para el uso de *autoconf* y *automake* simultáneamente. Facilita la personalización en proyectos que requieran macros propias. [12]

autoheader

autoheader crea una plantilla con un conjunto de directivas *#define* que pueden emplearse desde los programas en C. Esta obtiene como resultado final un solo lugar con el nombre de la aplicación y su versión, facilitando la mantención y la liberación de nuevas versiones. [12]

autoconf

autoconf es una herramienta que permite construir paquetes de programas de una manera más portable. Provee una serie de pruebas que se realizan en el sistema donde se instalará para determinar sus características antes de ser compilado, de tal forma que el código fuente de un programa puede adaptarse en mejor manera a los diferentes sistemas.[12]

Permite generar un *script*, llamado *configure*, que es el encargado de realizar las comprobaciones para determinar la disponibilidad y ubicación de todos los requisitos para la construcción exitosa de un programa. Cuando se encuentra bien construido, permite instalar y desinstalar muy fácilmente las aplicaciones. [12]

automake

automake es una herramienta que permite generar en forma automática archivos *Makefile*, de acuerdo a un conjunto de estándares, simplificando el proceso de organización de las distintas reglas así como proveyendo funciones adicionales que permiten un mejor control sobre los programas. [12]

La utilidad *make* es ampliamente usada en el desarrollo de aplicaciones, ya que a partir de un archivo, llamado *Makefile*, que contiene reglas de dependencia es capaz de determinar las acciones a seguir, comúnmente determinar que programas deben compilarse para obtener la aplicación. *automake* puede

proveer los archivos que son estándares en cada aplicación, y salvo que se desee realizar algún cambio, es posible trabajar transparentemente con lo que sugiera. [12]

1.8.6 Lenguaje de programación

Lenguaje C++

Bjarne Stroustrup crea una versión experimental denominada "C with Classes" hacia 1979, con la intención de proporcionar una herramienta de desarrollo para el kernel Unix en ambientes distribuidos. En particular, él considera que ciertas características del lenguaje "Simula" (notablemente su orientación a objetos) son útiles en desarrollos de software complejos. Es así como decide extender el lenguaje C.

En 1983 el lenguaje se rebautiza como C++ y en 1985 Stroustrup publica la primera edición del libro "The C++ Programming Language" que sirvió de estándar informal y texto de referencia. Posteriormente el lenguaje fue estandarizado (ISO C++) y paralelamente son publicadas la segunda y tercera ediciones de "The C++ Programming Language" de modo tal que reflejan estos cambios.

Desde sus inicios, C++ intentó ser un lenguaje que incluye completamente al lenguaje C (quizá el 99% del código escrito en C es válido en C++) pero tiene algunas cuestiones más pulidas como un control más estricto en el manejo de tipos de datos, y otras características que ayudan a la programación libre de errores, tales como: Programación Orientada a Objetos (POO), sobrecarga de operadores, *templates* o plantillas. Es un lenguaje de programación de propósito general, especialmente indicado para la programación de sistemas por su flexibilidad y su potencia.

Capítulo 2 Solución Propuesta

Introducción

En este capítulo se propone la solución al problema en cuestión, con el propósito de satisfacer el objetivo de la investigación. Además se expone consideraciones técnicas generales para llevar a cabo el desarrollo de la solución.

2.1 Solución Propuesta.

El ya mencionado módulo de AGDH cuenta con la implementación de un subsistema de balance de carga, que se ubica dentro del paquete *App* (Figura 16) que es el núcleo de la aplicación y controla el funcionamiento de todo el módulo.

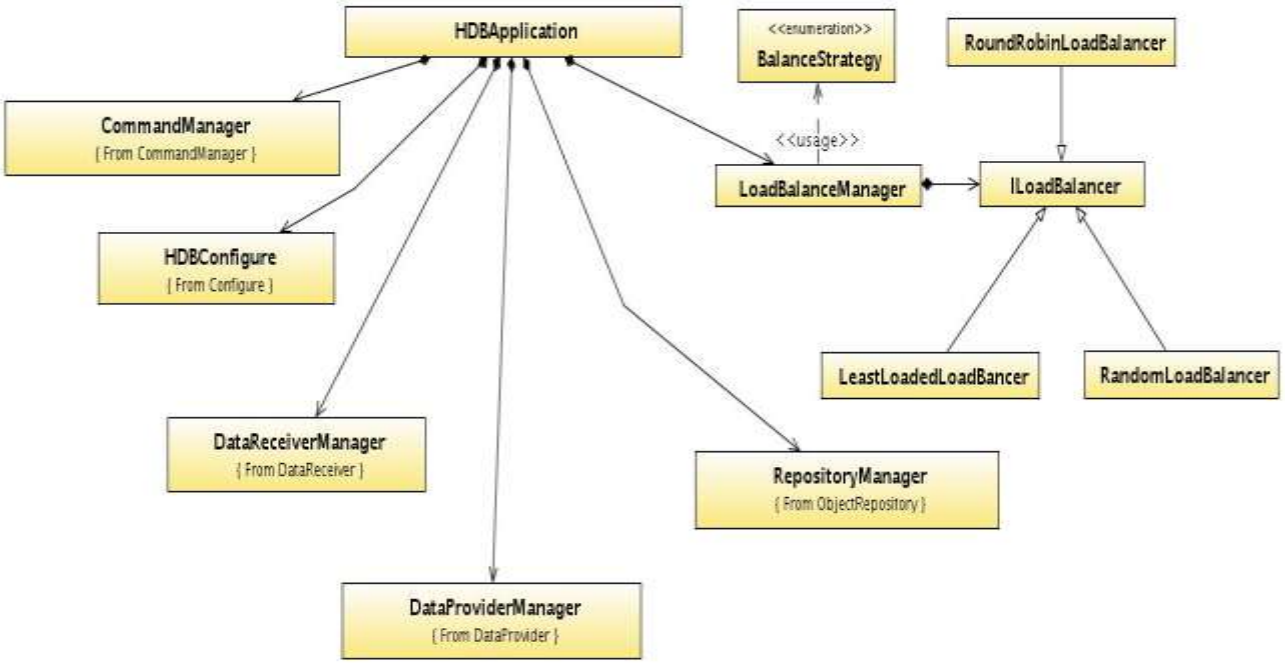


Figura 16. Diagrama de clases del paquete App. [13]

La siguiente figura (Figura 17) muestra las clases que componen el subsistema de balance de carga, evidenciando el uso de patrón Estrategia.

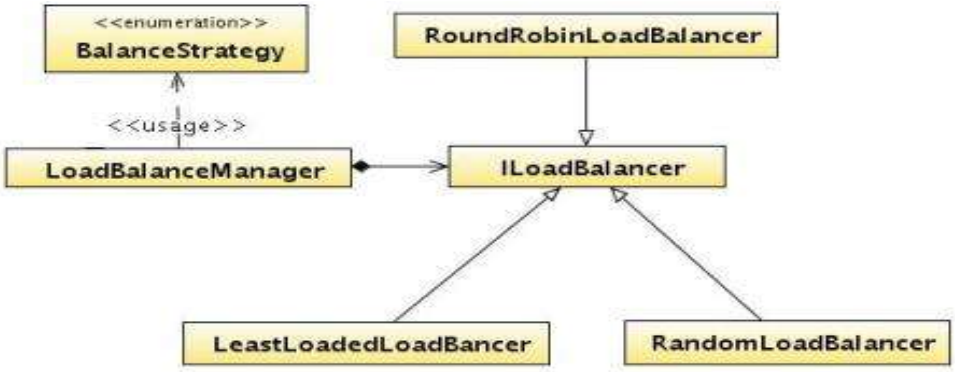


Figura 17. Clases que componen el balance de carga.

A continuación se describe el objetivo de estas clases.

***LoadBalanceManager*:** Clase que controla el balance de carga, proporciona la implementación de cada una de las estrategias para el mismo. [13]

***ILoadBalancer*:** Interfaz de balance de carga, define las responsabilidades de cada una de las implementaciones de las estrategias. [13]

***RandomLoadBalancer*:** Implementa la estrategia **aleatoria** de balance de carga. [13]

***RoundRobinLoadBalancer*:** Implementa la estrategia **cíclica** de balance de carga. [13]

***LeastLoadedLoadBalancer*:** Implementa la estrategia de **menor carga** de balance de carga. [13]

2.1.1 La estrategia de balance basada en Lógica Difusa.

Inicialmente se propone una nueva estrategia, basada en Lógica Difusa usando reglas para la toma de decisión. La razón por la que se propone esta estrategia basada en lógica difusa es porque su uso para el control (plantas industriales complejas, etc.) ha sido explotado anteriormente obteniendo excelentes resultados, pues permite una mayor precisión al momento de la toma de decisiones en el manejo de sistemas con cierto grado de incertidumbre. La principal ventaja de utilizar la Lógica Difusa está en que permite plantear el problema en los mismos términos en los que lo haría un experto humano y busca la solución no perfectamente definida por medio de un planteamiento matemático muy exacto, si no que primero razona empleando la inexactitud.

Esta nueva estrategia de acuerdo a lo investigado podría traer mejoras al balance de carga ya implementado. Permitiría aprovechar al máximo la capacidad de los hilos de procesamiento disponibles teniendo en cuenta las prioridades que se le debe dar a cada tipo de dato de acuerdo a la complejidad que implique su procesamiento.

El trabajo con las reglas difusas brinda facilidades de uso a la hora de aplicar esta estrategia, ya que no hay que tener conocimientos de programación para crearlas, sólo saber su estructura. Las reglas son de fácil interpretación y contienen el conjunto de acciones a realizar en función del estado. Los mecanismos de inferencia utilizan estas reglas para realizar el proceso de razonamiento para estimar la salida en función de la entrada y junto a otros componentes conforman un controlador difuso.

Un controlador difuso incluye 5 componentes (Figura 18): fuzzificación, base de reglas, funciones de membresía, máquina de inferencia fuzzy y defuzzificación.

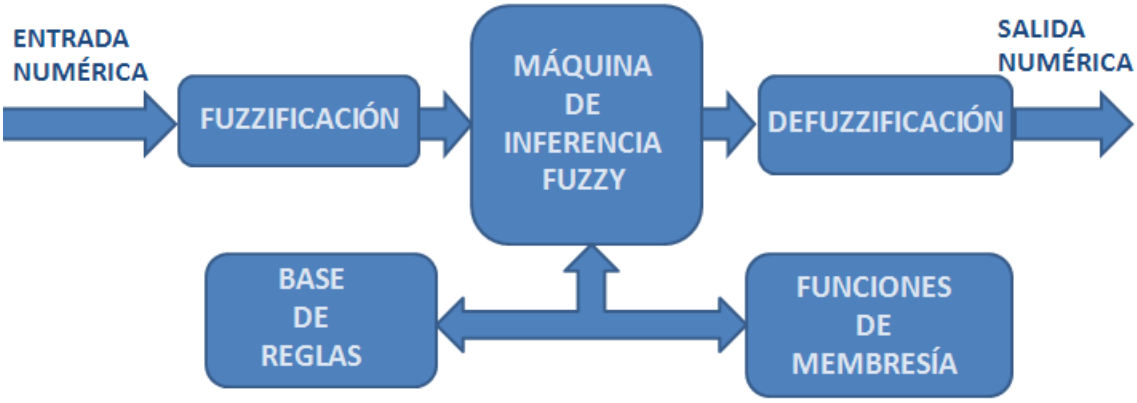


Figura 18. Arquitectura del modelo Fuzzy Logic.

Fuzzificación. Es la interfaz de entrada que mapea una entrada numérica a un conjunto fuzzy para que este pueda ser correspondido con las premisas de las reglas fuzzy definidas en la base de reglas específicas para la aplicación. [4]

Base de Reglas. Contiene un conjunto de reglas de fuzzificación if-then que define las acciones del controlador en términos de variables lingüísticas y funciones de membresía de términos lingüísticos. [5]

Funciones de Membresía. La función de membresía o pertenencia de un conjunto fuzzy consiste en un conjunto de pares ordenados si la variable es discreta, o una función continua si no lo es. El valor indica el grado en que la variable está incluida en el concepto representado por la etiqueta. Para la definición de estas funciones de pertenencia se utilizan convencionalmente ciertas familias de forma estándar, por coincidir con el significado lingüístico de las etiquetas más utilizadas. [4]

Máquina de Inferencia Fuzzy. Aplica el mecanismo de inferencia al conjunto de reglas en la base de reglas fuzzy para producir un conjunto de salida. [4]

Defuzzificación. Es un mapeador de salida que convierte el conjunto fuzzificado de salida, a una salida crisp. Basada en la salida crisp, el controlador fuzzy puede manejar el sistema bajo control. [4]

2.1.2 Consideraciones Técnicas Generales.

La selección de las herramientas, metodología, sistema operativo, etc. para el desarrollo, se lleva a cabo teniendo en cuenta lo utilizado para desarrollar el subsistema de balance de carga especificadas en el documento de Especificación de Requerimientos de Software de Balance de Carga del módulo de Almacenamiento y Gestión de Datos Histórico [14], además de las ventajas y características de las mismas expuestas en el capítulo anterior para poder lograr así la compatibilidad de la solución propuesta.

A partir de lo anterior se decidió usar como sistema operativo Linux en su distribución Debian Lenny, usando RUP como metodología de desarrollo, UML para el modelado y Visual Paradigm de herramienta CASE, C++ como lenguaje de programación con automake para la codificación y compilación.

2.2 Biblioteca para el trabajo con lógica difusa.

Durante la presente investigación se llevó a cabo el estudio de varias bibliotecas que por sus características y funciones podrían facilitar el desarrollo de la solución propuesta. Por sus ventajas sobre las demás, su llegó a la conclusión de que FFLL y jFuzzyQt se ajustan más a nuestras necesidades.

2.2.1 FFLL

Anteriormente se menciona la biblioteca FFLL para el trabajo con la lógica difusa. La IEC ha publicado un estándar para la Programación del Control Difuso (Fuzzy Control Programming) (IEC 131-7 61).Desafortunadamente, este estándar no está disponible gratuitamente y se debe comprar. Sin embargo, el IEC 61 131-7 dispone de una norma que especifica un estándar: FCL, que se utiliza para describir los modelos de lógica difusa. FFLL es capaz de cargar archivos que cumplan este estándar. El uso de este brinda una serie de ventajas, como es el caso configurar o personalizar el fichero sin tener que recompilar el código de la aplicación. Esta biblioteca tiene una desventaja de uso para el desarrollo, ya que después de haber tenido contacto con Michael Zamzinski, colaborador de la publicación IA Programming Wisdom -específicamente del epígrafe 2.8 An Open-Source Fuzzy Logic Library- hizo saber que esta biblioteca no ha sido compilada en la distribución del sistema operativo (Linux/Debian Lenny) sobre el cual se propone desarrollar, además de que no ha sido actualizada desde el 2004. Por lo que entonces se propone el uso de la biblioteca jFuzzyQt que es un clon de la misma.

2.2.2 jFuzzyQt

Esta biblioteca, al igual que FFLL, puede cargar ficheros que cumplan con el estándar FCL. Su uso brinda una serie de ventajas sobre la anterior, pues está compuesta por funciones que facilitan el uso del fichero FCL que para la FFLL no estaban disponibles, como es el caso de que a través de esta se pueden cargar diversas variables de salida, varios bloques de reglas por separado, varios *Function Block* todo esto dentro del mismo fichero. La última actualización de JFuzzyQt es de este año y se implementa sobre el *core* de Qt Para la defuzzificación cuenta con la implementación de dos métodos: COG (Centro de Gravedad) y COGS (Centro de Gravedad para Singletons) que en el siguiente epígrafe se especifica dónde y cómo se coloca dentro del fichero para su uso. Esta biblioteca es de fácil uso y entendimiento, y es de código abierto, una de sus ventajas fundamentales.

2.3 FCL

2.3.1 Descripción del sistema borroso usando FCL.

```
FUNCTION_BLOCK nombre_controlador


VAR_INPUT
    nombre_variable_entrada tipo_dato;
. . .
END_VAR


VAR_OUTPUT
    nombre_variable_salida tipo_dato;
. . .
END_VAR
```

```
VAR
        nombre_variable_local tipo_dato;
. . .
END_VAR


FUZZIFY nombre_variable_entrada
     TERM nombre_función_pertenencia:= (x1, y1), (x2, y2),. . .;
            . . .
END_FUZZIFY


DEFUZZIFY nombre_variable_salida
       METHOD: método_defuzzificación;
END_DEFUZZIFY


RULEBLOCK nombre
      AND: MIN;
      ACCU: MAX;
      RULE 0: …;
      RULE 1: …;
END_RULEBLOCK
END_FUNCTION_BLOCK
```

2.3.2 Declaración de variables

Las posibles variables de entrada para el desarrollo de nuestra estrategia serán el estado de los buffer y el tipo de estructura. La variable de salida será buffer que da el hilo de procesamiento al que se le va a colocar los elementos.

Una vez declaradas las variables de entrada y las de salida se deben definir las funciones de pertenencia. FCL se refiere a función de pertenencia como TERM. Los puntos que definen a las funciones de pertenencia se declaran en pares (x, m(x)). El valor x tendrá un valor perteneciente al rango de la variable, mientras que m(x) tendrá un valor entre 0 y 1.

Tipo de estructura

Es el tipo de dato que se va a procesar para ser almacenado. Tiene información usada para darle así una prioridad o un tratamiento diferente a la hora de tomar una decisión. Por tanto, la variable tipo de estructura es definida en el conjunto fuzzy como: {Alarma, Evento, Variable, Bitácora}.

Alarma: Las alarmas identifican eventos que indican mal funcionamiento del sistema, los cuales podrían conllevar a catástrofes. [2]

Evento: Los eventos, se pueden definir como ocurrencias que pueden ser importantes para el análisis del comportamiento y seguridad del sistema. [2]

Variable: Variables del sistema, sean éstas variables de campo, memoria o calculadas. [2]

Bitácora: Permiten a los operarios documentar situaciones de operación que podría ser interesante consultar por él o por los demás operadores en un futuro. [2]

Estado de buffer

Esta variable es la que da el estado de los buffers de acuerdo a la cantidad de elementos que tengan, por lo que va a estar definida en el conjunto fuzzy como: {Vacío, Casi_Vacío, Casi_Lleno, Lleno}.Al igual que el tipo de estructura, se tiene en cuenta para colocar la carga, cumpliendo con las reglas establecidas. La variable es definida para cada uno de los buffers con que se va a disponer, es decir, hay un estado de

buffer_1, buffer_2... buffer_n y así sucesivamente con la cantidad de buffers con que dispone el sistema. De esta forma se comprueba el estado de cada uno de ellos.

Buffer

Buffer es la variable de salida, y tiene tantas posibles salidas como hilos con los que se dispone para el balanceo. Es la variable que se va tener en cuenta para dar el resultado de la decisión, ya que lo que se decide es en qué buffer colocar el elemento de acuerdo a las demás variables. Esta variable puede ser modificada a partir de las especificaciones que tenga el sistema.

2.3.3 Método de Defuzzificación

Una vez fuzzificadas las entradas, el motor de inferencia se encargará de aplicar el *modus ponens* difuso a las entradas fuzzificadas y combinar las salidas de cada regla para generar una salida global. Como resultado de este proceso se obtendrá un área y no un valor concreto de la variable salida.

El siguiente paso es indicar cómo se quiere defuzzificar la salida. Los métodos de defuzzificación están descritos en la siguiente tabla (Tabla 1) y se emplean sobre el área obtenida:

Tabla 1 Métodos de defuzzificación.

Palabra Clave	Explicación
COG	Centro de Gravedad
COGS	Centro de Gravedad para Singletons
COA	Centro de Áreas
RM	Máximo por la Derecha

Existen trabajos relacionados con el empleo de los diferentes métodos de defuzzificación, por lo que para un mejor entendimiento de estos métodos se recomienda el estudio de los mismos en la siguiente bibliografía: [19] [20] [21] [22] [23].

2.3.4 Definición de Reglas

Por último, se rellena el bloque de reglas que se deben seguir para llevar a cabo el razonamiento. Las reglas pueden definirse de las dos siguientes maneras:

RULE 0: IF Alarma AND Casi_Vacio THEN Buffer_n;

O

RULE 0: IF Tipo_de_estructura IS Alarma AND Estado_buffer_n IS Casi_Vacio THEN Buffer IS Buffer_uno);

En el primer caso se asume que la primera condición se refiere a la primera variable de entrada declarada, y que la segunda condición se refiere a la segunda variable de entrada declarada. En el segundo caso, todo se indica explícitamente.

Se pide crear reglas con todas las combinaciones posibles entre las variables de entrada, para dar por finalizado el fichero fcl.

2.4 Reglas del negocio

- El fichero de reglas debe cumplir el estándar FCL y las condiciones de la biblioteca jFuzzyQt en cuanto a su estructura explicadas anteriormente.

- Las reglas difusas deben contener todas las combinaciones posibles según las variables de entrada, y así poder obtener salidas válidas para todos los casos o la mayoría de estos.

2.5 Modelo conceptual.

2.5.1 Descripción del Modelo conceptual.

Este modelo servirá como referencia para la comprensión de todos los términos manejados en la elaboración de la solución propuesta.

Se representan (Figura 19) a continuación los conceptos y sus relaciones en la solución específica:

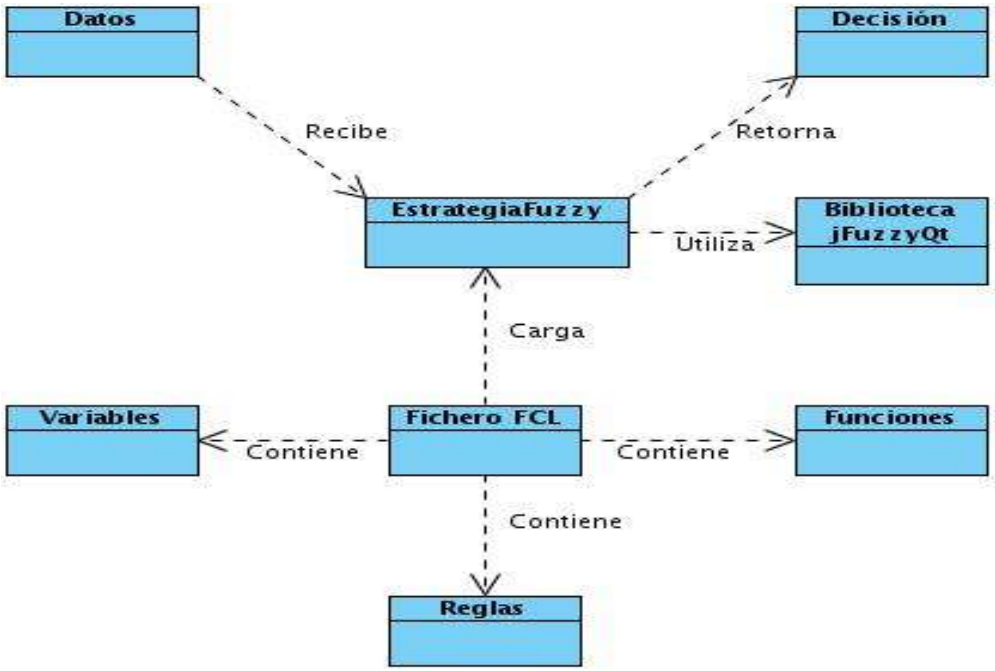


Figura 19. Modelo conceptual de la solución específica.

En el siguiente diagrama (Figura 20) se representa una vista general de la solución específica una vez ubicada en su entorno:

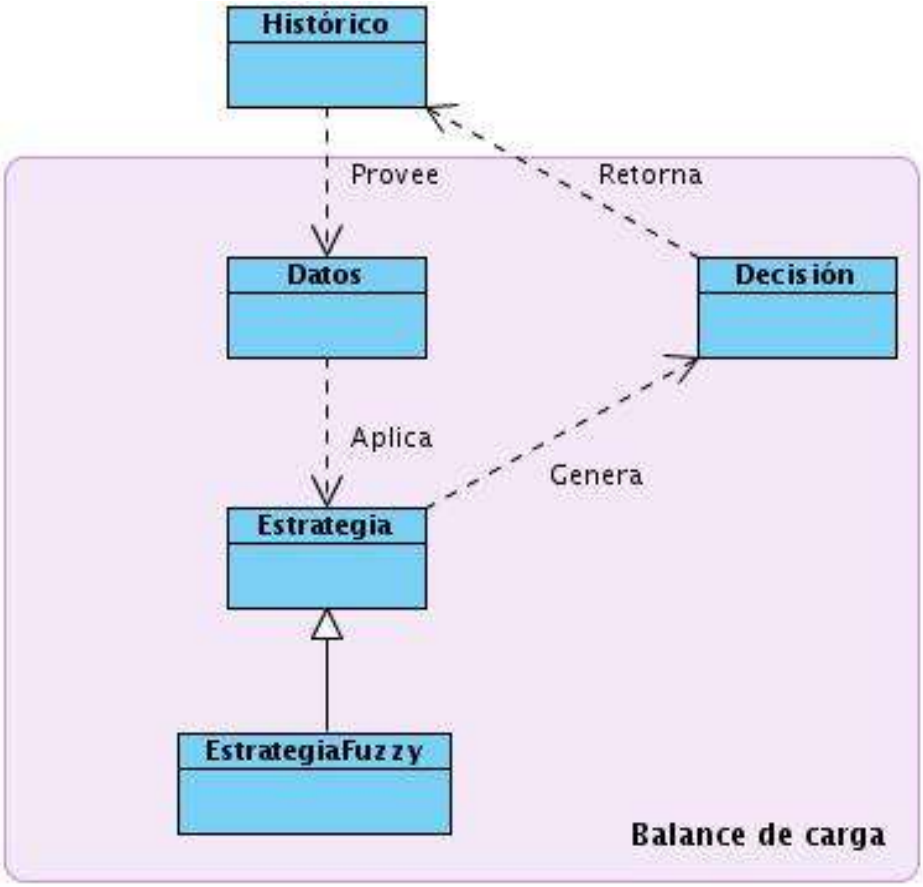


Figura 20. Modelo conceptual general.

2.5.2 Glosario de términos del modelo conceptual.

Histórico: Módulo de almacenamiento y gestión de datos que provee los datos.

Datos: Es la expresión general que describe las características de las entidades que brinda el Histórico sobre las cuales opera la EstrategiaFuzzy.

Estrategia: Es el proceso seleccionado mediante el cual se espera tomar una decisión.

EstrategiaFuzzy: Proceso mediante el cual la lógica difusa basada en reglas logra tomar decisiones.

Decisión: Conjunto de alternativas disponibles tenidas en cuenta para dar el resultado de la aplicación de la estrategia.

Biblioteca jFuzzyQt: Biblioteca que utiliza la EstrategiaFuzzy para cargar el fichero FCL.

Fichero FCL: Fichero que está formado por variables, funciones y reglas, cargado por la biblioteca jFuzzyQt y que utiliza la estrategia para llegar a la decisión.

Variables: Representan los datos de entrada y salida que contiene el fichero FCL.

Funciones: Función de defuzzificación que contiene el fichero FCL.

Reglas: Normas que deben cumplir las variables de entrada para retornar una variable de salida válida.

Balance de Carga: El proceso de aplicar la estrategia a los datos.

2.6 Requisitos

2.6.1 Requisitos Funcionales.

Los Requisitos Funcionales se definen como las capacidades o condiciones que el producto debe cumplir y no varían, sin importar con qué propiedades o cualidades se relacionen.

La estrategia debe cumplir los siguientes requisitos funcionales:

1. Crear fichero de configuración.
   1.1 Definir variables de entrada y salida.
   1.2 Definir método de defuzzificación.
   1.3 Definir reglas.
2. Crear modelo.
3. Cargar fichero de configuración.
4. Fuzzificar variables de entrada.

5. Desarrollar inferencia.
6. Defuzzificar.
7. Devolver variable de salida.

2.6.2 Requisitos No Funcionales.

Los Requisitos No Funcionales, se definen como propiedades o cualidades que el producto debe tener.

Los Requisitos No Funcionales con que debe contar la estrategia propuesta son los ya especificados en el documento Especificación de Requerimientos de Software para Balance de Carga del módulo Base de Datos de Históricos [14].

Capítulo 3 Análisis de Resultados

Introducción

En este capítulo se lleva a cabo un análisis de los resultados de las pruebas realizadas a la estrategia. Además se hace una comparación contra las estrategias existentes para llegar a conclusiones concretas sobre el funcionamiento de la propuesta.

3.1 FuzzyLoadBalancer en el subsistema de balance de carga.

En el capítulo anterior, la Figura 13 muestra el diagrama de clases del subsistema de balance de carga del módulo de AGDH. En este, se encuentran representadas cada una de las estrategias con las que cuenta el balance de carga actualmente, además de las clases que controlan el balance en general. En la siguiente figura (**Figura 21**) se muestra entonces un nuevo diagrama de clases que incluye la clase FuzzyLoadBalancer, estrategia propuesta, que implementa la estrategia de lógica difusa basada en reglas. Por lo tanto, este sería el diagrama de clases del balance de carga una vez incorporada la nueva estrategia.

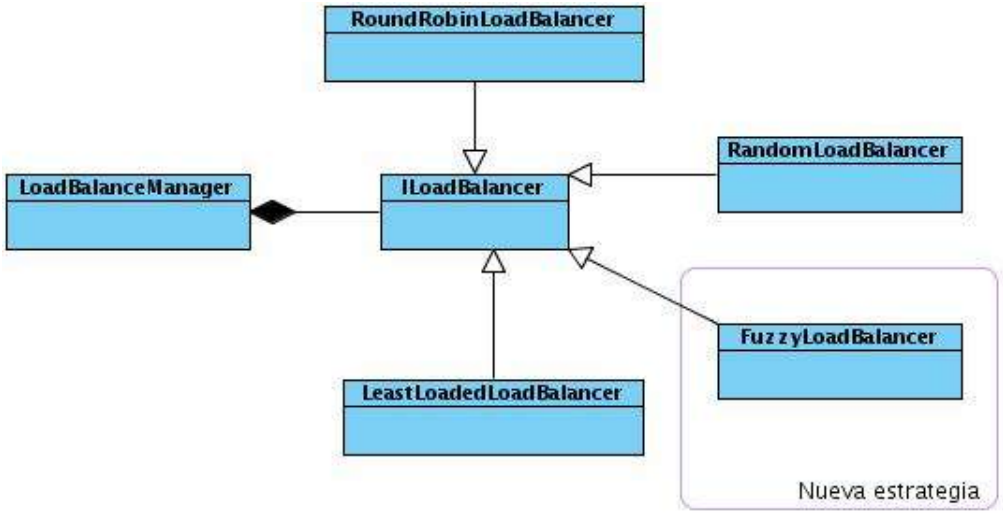


Figura 21. Diagrama de clases del Balance de Carga.

Para el satisfactorio acople de la estrategia se deben hacer una serie de cambios dentro de la clase *LoadBalanceManager* -ya que esta es la que controla todo el balance de carga- y en la clase *ILoadBalancer*, que es la interfaz y la que define la responsabilidades de las estrategias . El cambio consiste en agregarles el tipo de estructura (Alarma, Evento, Variable, Bitácora) ya que la nueva estrategia lo tiene en cuenta, no así en las anteriores estrategias implementadas.

Específicamente, este tipo de estructura es agregado cuando se declaran dichas clases en el siguiente método:
*Void setStates (StatesVector * currentStates,* ***unsigned int * currentDataType****);*
Este método anteriormente sólo asignaba el estado de los objetos y con el nuevo parámetro también provee el tipo de estructura; quedando como sigue: *currentStates* contiene la información del estado de los objetos y ***currentDataType*** contiene la información del tipo de estructura.

3.1.1 FuzzyLoadBalancer

La clase FuzzyLoadBalancer sigue la misma estructura de las demás estrategias.

```
class FuzzyLoadBalancer : public ILoadBalancer
{
public:

    /**
     *  Constructor de la clase.
     */
    FuzzyLoadBalancer();

    /**
     * @brief Destructor de la clase.
     */
    virtual ~FuzzyLoadBalancer();

    /**
     *  Método que devuelve el id del objeto a procesar el volumen de
     * datos recibidos. Aplica un algoritmo de lógica difusa.
     */
    unsigned int object();
};
```

Figura 22. Estructura de la clase FuzzyLoadBalancer.

Esta estrategia, a diferencia de las demás, incluye la biblioteca jFuzzyQt, descrita anteriormente.

```
#include "ILoadBalancer.h"
#include <LoadBalancer/jFuzzyQt/include/jfuzzyqt.h>
#include <iostream>
```

Figura 23. Dependencias de la clase FuzzyLoadBalancer

En la implementación de las estrategias, el método *unsigned int object ();* es el que devuelve el resultado de aplicar la misma, por lo que se implementa de forma diferente en cada una de ellas. En el caso específico de la estrategia FuzzyLoadBalancer es en este método donde se hace uso de la biblioteca a partir de cuyas funciones, se crea un modelo y se carga el fichero FCL. También se dan valores a las variables de entrada, y a partir de estos datos de entrada y las reglas del modelo cargadas se evalúa el modelo y se obtiene la salida, que corresponde con el id del buffer en el que se colocan los datos recibidos.

3.2 Herramienta de desarrollo y evaluación (QBalanceo).

Para evaluar el resultado de la estrategia Fuzzy acoplada a las clases del balance ya implementadas y comprobar que sigue el patrón seleccionado, se crea una pequeña aplicación, QBalanceo (nombre del proyecto) que simula el funcionamiento del balance de carga que se encuentra en el módulo de AGDH. Para ello, se reutiliza el código del balance de carga implementado para este módulo, incluyendo los cambios pertinentes explicados anteriormente.

El objetivo de esta aplicación es mostrar visualmente el trabajo de la estrategia Fuzzy, aunque contiene además todas las estrategias ya implementadas, lo que la convierte en una herramienta de desarrollo y evaluación (*stub*) tanto de las ya implementadas como de posibles mejoras o adiciones en el futuro. Esta herramienta es necesaria ya que el balance de carga está incluido internamente en el módulo y no hay forma de visualizar o monitorear este proceso. Además, la velocidad a la que esto se realiza es mucha, pues se procesan grandes cantidades de datos en milisegundos, y puede ser imperceptible a simple vista.

La aplicación es desarrollada sobre Qt ya que es totalmente orientado a objetos, fácil de usar, extensible y multi-plataforma (soportada en Windows, Unix y derivados). Cuenta con la implementación de una serie

de clases que muestra el diagrama representado en el Anexo 1.

La herramienta posee una serie de componentes visuales que permiten introducir datos necesarios para probar el funcionamiento de las estrategias. Estos datos son: estrategia que se desea probar, datos que se van a introducir, retardo (carga adicional para los buffer) y tipo de estructura. El proceso puede llevarse a cabo de forma manual o automática. Para realizarlo de forma manual se cuenta con un botón llamado *Generar* que realiza el proceso. Y para realizarlo de forma automática se introduce un tiempo en milisegundos. Este determina cada cuanto tiempo se van a generar los datos para guardarlos en la BD realizando el balance de carga. Además de otro tiempo que es el tiempo de duración de la prueba automática. El Anexo 5 muestra la interfaz de esta herramienta.

3.2.1 Descripción de las principales clases de QBalanceo.

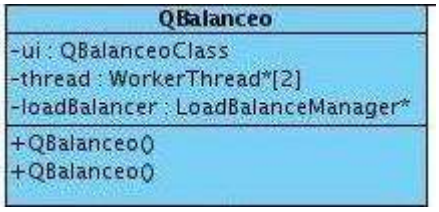


Figura 24. Clase QBalanceo.

Tabla 2. Descripción QBalanceo.

Nombre: QBalanceo	
Controladora	
Atributo	**Tipo**
QBalanceoClass	ui
thread	WorkerThread *
loadBalancer	LoadBalanceManager *
Para cada responsabilidad:	
Nombre:	QBalanceo()
Descripción:	Constructor de la clase
Nombre:	QBalanceo()
Descripción:	Destructor de la clase

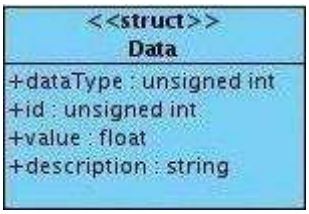


Figura 25. Clase Data.

Tabla 3. Descripción de Data.

Nombre: Data	
Entidad	
Atributo	**Tipo**
dataType	unsigned int
id	unsigned int
value	float
description	string
Para cada responsabilidad:	
Nombre:	
Descripción:	

Figura 26. Clase DataAccess.

Tabla 4. Descripción de DataAccess.

Nombre: DataAccess
Controladora

Atributo	**Tipo**
pqConn	connection*
tableName	string
Para cada responsabilidad:	
Nombre:	DataAccess()
Descripción:	Constructor por defecto de clase.
Nombre:	DataAccess(string ip, string dbname, string user, string passwd, string tName)
Descripción:	Constructor con parámetros de la clase.
Nombre:	DataAccess()
Descripción:	Destructor.
Nombre:	connectToDb(string ip, string dbname, string user, string passwd)
Descripción:	Realiza la conexión con la BD.
Nombre:	insert (Data * data)
Descripción:	Inserta un dato en la BD.
Nombre:	query(int id)
Descripción:	Devuelve un dato a partir de su id.
Nombre:	update(Data * data)
Descripción:	Actualiza el dato.
Nombre:	trunc()
Descripción:	Borra el contenido de la tabla.

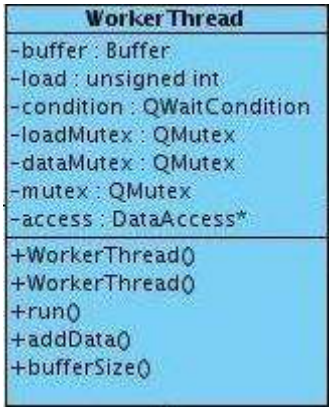


Figura 27. Clase WorkerThread.

Tabla 5. Descripción de WorkerThread.

Nombre: WorkerThread	
Controladora	
Atributo	**Tipo**
buffer	Buffer
load	unsigned int
condition	QWaitCondition
loadMutex	QMutex
dataMutex	QMutex
mutex	QMutex
access	DataAccess *
Para cada responsabilidad:	
Nombre:	WorkerThread()
Descripción:	Constructor de la clase.
Nombre:	WorkerThread()
Descripción:	Destructor de la clase.
Nombre:	run()
Descripción:	Insertar en la BD un dato y actualiza su estado.
Nombre:	addData(Data* data)
Descripción:	Adiciona un dato al buffer.
Nombre:	setLoad(int _load)
Descripción:	Le proporciona un retardo (carga adicional) al hilo.
Nombre:	bufferSize()
Descripción:	Retorna el tamaño del buffer.

Figura 28. Clase LoadBalanceManager.

Tabla 6. Descripción de LoadBalanceManager.

Nombre: LoadBalanceManager	
Controladora	
Atributo	**Tipo**
loadBalancer	ILoadBalancer *
Para cada responsabilidad:	
Nombre:	LoadBalanceManager(BalanceStrategy strategy)
Descripción:	Constructor de la clase, tiene una estrategia por defecto.
Nombre:	LoadBalanceManager()
Descripción:	Destructor de la clase.
Nombre:	setBalanceStrategy(BalanceStrategy strategy)
Descripción:	Cambia de estrategia.
Nombre:	object()
Descripción:	Retorna el resultado de la aplicación de la estrategia.
Nombre:	setStates(StatesVector * currentStates , unsigned int * currentDataType)
Descripción:	Asigna una estructura con el estado de los buffer y el tipo de dato.

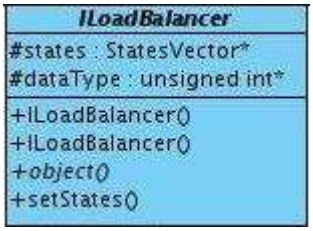


Figura 29. Clase ILoadBalancer.

Tabla 7. Descripción de ILoadBalancer.

Nombre: ILoadBalancer	
Interfaz	
Atributo	**Tipo**
states	StatesVector *
dataType	unsigned int *
Para cada responsabilidad:	

Nombre:	ILoadBalancer()
Descripción:	Constructor de la clase.
Nombre:	ILoadBalancer()
Descripción:	Destructor de la clase.
Nombre:	object()
Descripción:	Retorna el resultado de la aplicación de la estrategia. (método virtual)
Nombre:	setStates(StatesVector * currentStates, unsigned int * currentDataType)
Descripción:	Asigna una estructura con el estado de los buffer y el tipo de dato.

Figura 30. Clase LeastLoadedLoadBalancer

Tabla 8. Descripción de LeastLoadedLoadBalancer.

Nombre: LeastLoadedLoadBalancer		
Controladora		
Atributo		**Tipo**
Para cada responsabilidad:		
Nombre:	LeastLoadedLoadBalancer()	
Descripción:	Constructor de la clase.	
Nombre:	LeastLoadedLoadBalancer()	
Descripción:	Destructor de la clase.	
Nombre:	object()	
Descripción:	Retorna el id del buffer donde se coloca la información.	

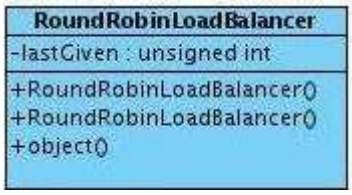


Figura 31. Clase RoundRobinLoadBalancer.

Tabla 9. Descripción de RoundRobinLoadBalancer.

Nombre: RoundRobinLoadBalancer	
Controladora	
Atributo	**Tipo**
lastGiven	unsigned int
Para cada responsabilidad:	
Nombre:	RoundRobinLoadBalancer()
Descripción:	Constructor de la clase.
Nombre:	RoundRobinLoadBalancer()
Descripción:	Destructor de la clase.
Nombre:	object()
Descripción:	Retorna el id del buffer donde se coloca la información.

Figura 32. Clase FuzzyLoadBalancer.

Tabla 10. Descripción de FuzzyLoadBalancer.

Nombre: FuzzyLoadBalancer	
Controladora	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	FuzzyLoadBalancer()
Descripción:	Constructor de la clase.

Nombre:	FuzzyLoadBalancer()
Descripción:	Destructor de la clase.
Nombre:	object()
Descripción:	Retorna el id del buffer donde se coloca la información.

Figura 33. Clase RandomLoadBalancer.

Tabla 11. Descripción de RandomLoadBalancer.

Nombre: RandomLoadBalancer		
Controladora		
Atributo		**Tipo**
Para cada responsabilidad:		
Nombre:	RandomLoadBalancer()	
Descripción:	Constructor de la clase.	
Nombre:	RandomLoadBalancer()	
Descripción:	Destructor de la clase.	
Nombre:	object()	
Descripción:	Retorna el id del buffer donde se coloca la información.	

3.2.2 Diagrama de secuencia.

La siguiente imagen (**Figura 34**) muestra el diagrama de secuencia correspondiente a la creación e inicialización del paquete LoadBalancer. Este recibe a través del constructor o del método setBalanceStartegy (strategy: BalanceStrategy) el tipo de estrategia a implementar, creando una de las cuatro posibles variantes (una vez colocada la estrategia Fuzzy): RandomLoadBalancer, RoundRobinLoadBalancer, LeastLoadedLoadBalancer o FuzzyLoadBalancer. Es una implementación del patrón de diseño Estrategia, lo cual permitirá en futuras versiones cambiar la estrategia en tiempo de ejecución a través de la configuración en caliente o el envío de comandos al módulo.

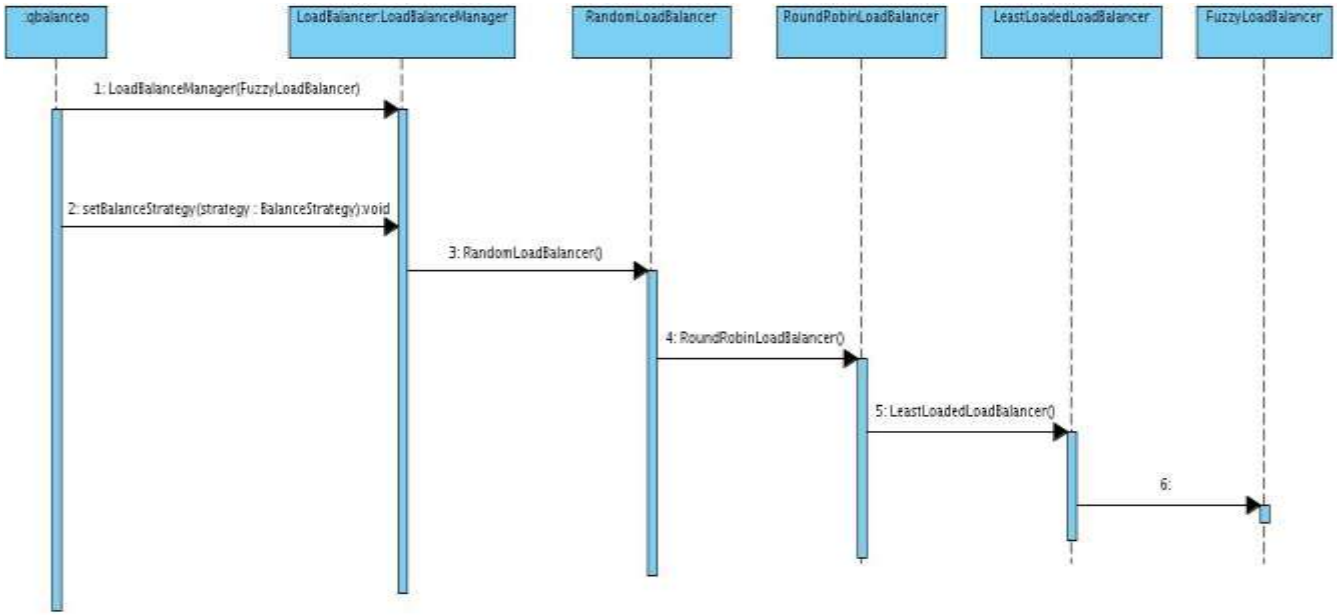


Figura 34. Diagrama de secuencia

3.2.3 Fichero FCL utilizado por FuzzyLoadBalancer en QBalanceo.

La estrategia FuzzyLoadBalancer carga un fichero FCL a partir de una función que brinda la biblioteca jFuzzyQt. En este fichero se encuentran declarados las variables de entrada y de salida, así como el método de defuzzificación y las reglas a utilizar. La estructura del mismo se encuentra descrita en el capítulo anterior.

Variables.

Las variables juegan un papel muy importante dentro del fichero y del modelo en general, ya que a través de ellas se indica la entrada y las reglas pueden realizar el proceso de razonamiento. En el capítulo anterior se describe cómo se declaran las variables, tanto de entrada como de salida.

Variables de entrada

En la estrategia FuzzyLoadBalancer se tienen en cuenta una serie de parámetros que son los que van a representar a las variables de entrada del fichero FCL utilizado. Como se menciona en el capítulo anterior estas variables de entrada van a ser: *Tipo_de_estructura, Estado_buffer_n.* cada una de ellas con su respectiva función de pertenencia.

El fichero que utiliza QBalanceo fue declarado como *ejemplo.fcl* y dentro del mismo se ubica *FUNCTION_BLOCK balance*, que contiene todas las variables y reglas. Cada variable tiene una declaración particular para probar la estrategia FuzzyLoadBalancer, que no necesariamente es la que tiene que ser utilizada una vez que se agrega la nueva estrategia al módulo. En este caso lo que cambia no es el nombre de las variables o el nombre de las etiquetas declaradas en el fichero, si no sus valores de pertenencia, que se adecuan a la situación a la que se aplique la estrategia.

Los tipos de estructuras no van a variar, ya que estos son los tipos reales con los que trabaja el módulo de AGDH. Para la aplicación sólo se van a tener en cuenta dos tipos de estructura: *Alarma* y *Variable*, ya que el *Evento* y la *Bitácora* son procesados de forma muy similar a la *Variable.*

A continuación, la declaración de la variable de entrada *Tipo_de_estructura* dentro del fichero *ejemplo.fcl*.

```
FUZZIFY Tipo_de_estructura
      TERM Alarma:= 1;
      TERM Variable:= 2;
END_FUZZIFY
```

La otra variable de entrada es el *Estado_buffer_n* que como se explica en el capítulo anterior se declara para cada uno de los buffer con que se dispone. En este caso específico sólo se van a utilizar dos buffer.

Declaración de la variable *Estado_buffer_n* dentro del fichero *ejemplo.fcl.*

```
FUZZIFY Estado_buffer_1
      TERM Vacio := 0;
      TERM Casi_Vacio := (0, 0) (100, 1) (250, 1) (500,0);
      TERM Casi_LLeno := (250, 0) (500, 1) (750, 0);
      TERM LLeno := (500, 0) (750, 1) (1000, 1) (1000, 0) ;
END_FUZZIFY


FUZZIFY Estado_buffer_2
```

```
      TERM Vacio := 0;
      TERM Casi_Vacio := (0, 0) (100, 1) (250, 1) (500,0);
      TERM Casi_LLeno := (250, 0) (500, 1) (750, 0);
      TERM LLeno := (500, 0) (750, 1) (1000, 1) (1000, 0) ;
END_FUZZIFY
```

Los rangos de valores para la función de pertenencia se llevan hasta 1000 para que cuando la aplicación lleve a cabo la visualización de lo hilos de procesamiento pueda ser perceptible.

Variables de salida

La variable de salida declarada para este fichero es *Buffer* que no es más que los hilos de procesamiento con los que se cuenta para la distribución del trabajo. QBalanceo utiliza dos hilos para la ubicación de la carga y un tercero que no se muestra en la aplicación, ya que sólo sería utilizado cuando los dos buffer disponibles estén completamente ocupados.

Declaración de la variable *Buffer* dentro del fichero *ejemplo.fcl.*

```
DEFUZZIFY Buffer
      TERM Buffer_uno := 1;
      TERM Buffer_dos := 2;
      TERM Buffer_tres := 3;
      METHOD : COGS;
      DEFAULT := 0;
END_DEFUZZIFY
```

Cada una de estas variables se encuentra representada gráficamente en los Anexos 2, 3 y 4.

Método de defuzzificación

El método de defuzzificación utilizado es COGS (Centro de Gravedad para Singletons) que es uno de los dos que implementa la biblioteca jFuzzyQt.

Reglas

Las reglas utilizadas en el fichero no son reglas creadas específicamente para el módulo, sólo son reglas

para probar el funcionamiento de la estrategia. Esto no quiere decir que no se puedan utilizar, pero no están basadas en casos reales del módulo. Una vez que se vaya a utilizar la estrategia, estas reglas pueden ser creadas sin complicación ya que son de fácil estructura.

3.3 Resultados.

En este epígrafe se tiene por objetivo tomar casos de prueba para evaluar el desempeño de la estrategia FuzzyLoadBalancer.

3.3.1 Estadísticas de las pruebas.

A continuación se llevan a cabo una serie de pruebas para analizar y valorar estadísticamente los resultados y poder llegar a conclusiones concretas sobre el funcionamiento de la estrategia y para así compararla así con las ya implementadas.

Para realizar estas pruebas se forzó a correr la aplicación con datos de iguales para cada estrategia, y verificar así la cantidad de buffer que se utiliza para cada caso. En las siguientes tablas se muestran los datos de las pruebas. Se llevó a cabo el control automático de la aplicación. Por lo que se prueba durante **1 minuto** y generando los datos cada **5000 milisegundos**. Cada buffer tiene un retardo adicional diferente en cada una de las pruebas, se decide probar para la misma carga, y para cargas diferentes.

Tabla 12. Prueba #1.

Nombre de la estrategia		Cantidad de datos a procesar	Cantidad de buffers utilizados	Retardo (carga adicional)	Tipo de estructura
1	Random	50	2	527027	Alarma
				121622	
2	Round Robin	50	2	527027	Alarma
				121622	
3	Least Loaded	50	2	527027	Alarma
				121622	
4	Fuzzy	50	1	527027	Alarma
				121622	

Tabla 13. Prueba #2.

Nombre de la estrategia		Cantidad de datos a procesar	Cantidad de buffers utilizados	Retardo (carga adicional)	Tipo de estructura
1	Random	50	2	297297 797297	Variable
2	Round Robin	50	2	297297 797297	Variable
3	Least Loaded	50	2	297297 797297	Variable
4	Fuzzy	50	1	297297 797297	Variable

Tabla 14. Prueba #3.

Nombre de la estrategia		Cantidad de datos a procesar	Cantidad de buffers utilizados	Retardo (carga adicional)	Tipo de estructura
1	Random	100	2	486486 486486	Alarma
2	Round Robin	100	2	486486 486486	Alarma
3	Least Loaded	100	2	486486 486486	Alarma
4	Fuzzy	100	2	486486 486486	Alarma

Tabla 15. Prueba #4.

Nombre de la estrategia		Cantidad de datos a procesar	Cantidad de buffers utilizados	Retardo (carga adicional)	Tipo de estructura
1	Random	100	2	500000 500000	Variable
2	Round Robin	100	2	500000 500000	Variable
3	Least Loaded	100	2	500000 500000	Variable
4	Fuzzy	100	1	500000 500000	Variable

3.3.2 Valoración de los resultados.

Con las pruebas realizadas anteriormente se puede comparar el funcionamiento de cada una de las estrategias.

Porciento de utilización de los buffer: Durante la realización de las pruebas se le presta especial atención al porciento de uso de los buffer.

En el caso específico de la Prueba #2. Se observa como todas las estrategias utilizan ambos buffer a un 17% aproximadamente. Mientras que Fuzzy puede asumir la carga en un solo buffer, aunque este trabaja a un 33%. El que Fuzzy haya trabajado a un porciento mayor no implica que sea menos eficiente si no que logró la utilización de un solo buffer. En el caso de las estrategias Random, Round Robin y Least Loaded tuvieron los dos buffer activos y la suma del porciento de uso de estos es el porciento que utilizó Fuzzy en un solo buffer. Trae como ventaja que se haga un eficiente uso de la capacidad de almacenamiento de los buffer. Que no se tengan activos todos los buffer a la menor capacidad, si uno solo es capaz de realizar la tarea.

Tipo de estructura: Fuzzy para cada tipo de estructura da un tratamiento diferente de acuerdo a lo planteado en las reglas, mientras que las demás estrategias nunca tienen en cuenta esto. Eso da la posibilidad de que se analice el tipo de estructura antes de ser procesado. Así se le da la ubicación

adecuada de acuerdo a prioridad que tenga cada tipo de estructura definido.

Estado de los buffer: Este criterio no se tiene en cuenta para la aplicación de las estrategias Round Robin, ni para Random. Solo Least Loaded que coloca los datos donde haya menos carga. Mientras que Fuzzy realiza un análisis del estado de los buffer para llevar a cabo una mejor distribución de la carga teniendo en cuenta además el tipo de estructura a procesar.

Según lo observado en las pruebas las estrategias Random y Round Robin no realizan un análisis del estado del buffer. El primero coloca los datos aleatoriamente en los dos buffer. Por otro lado Round Robin tampoco analiza este estado pero, trata de mantener la carga lo más pareja posible en cada buffer. Least Loaded tiene en cuenta este estado para colocar los datos en el buffer que tenga menor carga, pero no cuestiona la capacidad total del buffer.

Fuzzy por su parte siempre verifica el estado actual de cada buffer. Analiza además la posibilidad de colocarlo en uno u otro aprovechando al máximo la capacidad de almacenamiento total con la que dispone. Mediante este análisis puede determinar si tener activo o no el buffer, además de conocer cuando y donde se pueden ubicar los datos a procesar. Todo esto de acuerdo a la capacidad definida para cada uno y de cuando se considera vacio, casi lleno o lleno.

Uso de los buffer: Todas las estrategias utilizan siempre dos buffer, a diferencia de Fuzzy, que solo lo utiliza en caso de que sea necesario. Esto trae como ventaja una reducción del consumo de procesamiento de la aplicación al liberar hilos de procesamiento innecesarios, sin descuidar el principal objetivo de esta que es garantizar la persistencia de los datos.

Conclusiones Generales

La estrategia propuesta cumple con la misma estructura que las demás estrategias ya implementadas para el subsistema de balance de carga. Además, promueve la independencia, la facilidad de entender el diseño y la posibilidad de futuras extensiones ya que sigue el patrón Estrategia. El diseño e implementación de la estrategia posibilita un acople de esta al subsistema. El algoritmo de lógica difusa dota a la estrategia de la capacidad de tomar decisiones de acuerdo a la situación dada.

La herramienta de desarrollo y evaluación cumple con los parámetros necesarios para la realización de las pruebas. Estas pruebas, posibilitan la comparación de la estrategia propuesta con las existentes en el subsistema, en cuanto a porciento de utilización de los buffer, cantidad de buffer utilizados y el tipo de estructura procesado. Sirve además de base para futuros desarrollos de otras estrategias y su evaluación.

La estrategia propuesta hace un eficiente uso de la capacidad de almacenamiento de los buffer. Posibilita que se analice y se tenga en cuenta el tipo de estructura antes de ser procesado. Reduce el consumo de procesamiento al liberar hilos de procesamiento innecesarios, sin descuidar su principal objetivo que es garantizar la persistencia de los datos.

Recomendaciones

Para futuras mejoras de la investigación se hacen las siguientes recomendaciones:

- Acoplar la estrategia propuesta al subsistema de balance de carga del módulo de AGDH para que pueda ser utilizada por el mismo.
- Llevar a cabo el desarrollo de un editor para las reglas dentro del configurador del módulo para facilitar el proceso de creación de las reglas.

Referencias Bibliográficas.

1. **Betron, José Ignacio.** *Modelo de toma de decisiones y aprendizaje en sistemas multi-agentes.* Madrid: s.n., 1999.

2. **de la Horra Díaz, Abdelaziz.** *Desarrollo del subsistema de comunicación para el Módulo Base de Datos de Históricos del proyecto SCADA Nacional.* 2008.

3. **Bozyigita, M.** *"History-driven dynamic load balancing for recurring applications on network of workstations".* 2000.

4. **Luna Rosas, Francisco Javier, et al.** *Integrando mediante patrones de software una estrategia fuzzy-logic en un servicio de balanceo de carga bajo CORBA.* 2008.

5. **MAURICIO AGUDELO, O**. *Introducción a la lógica difusa.*

6. **AI Game Programming WISDOM** [Informe]

7. **Control** [Book Section] / auth. Juárez Herrero José M., Cañadas Martínez José J. and Marín Morales Roque.

8. **Sistema de Toma de Decisiones Para Videojuegos** [Informe] / aut. Ever Antonio Homer Reynoso Iriam Alberto González Boffill. - Ciudad de la Habana : [s.n.], 2009.

9. **Martínez, Joan.** *GNU/Linux.*

10. **S.Pressman, Roger. 2005.** *Ingeniería del Software.* La Habana: Félix Varela, 2005.

11. *http://es.wikipedia.org/.* (2010, Enero 19). Retrieved enero 28, 2010, from http://es.wikipedia.org/: http://es.wikipedia.org/wiki/Herramienta_CASE

12**.autotools: Herramientas para la creación de proyectos Open Sourse** [Informe] / aut. Caamaño Germán Poo. - 2004.

13**.SCADA Nacional, Diseño de la solución de Balance de Carga. Módulo Base de Datos de Históricos.** [Informe] / aut. de la Horra Ing. Abdelaziz. - 2009.

14. **Especificación de Requerimientos de Software para Balance de Carga del módulo Base de Datos de Históricos** [Informe].

Bibliografía.

1. **GNU/Linux.** [Report] / auth. Martínez Joan...
2. **Sistemas Expertos, Redes Bayesianas y sus aplicaciones. 2005.** [Report] / auth. Marín Illera Álvaro…
3. **autotools: Herramientas para la creación de proyectos Open** [Report] / auth. Poo Caamaño Germán. - 2004.
4. **Computación Paralela** [Book] / auth. GARCÍA O., CRUZ, R., Y CARRILLO, H… - 2003.
5. **DESARROLLO DE MÓDULO DE BASE DE DATOS HISTÓRICOS PARA SISTEMA SCADA** [Report] / auth. Gustabello Cogle Robby [et al.]. - 2008.
6. **Desarrollo Del subsistema de comunicación para el Módulo Base de Datos de Históricos del proyecto SCADA Nacional.** [Report] / auth. de la Horra Díaz Abedaliziz. - 2008…
7. **Especificación de Requerimientos de Software para Balance de Carga del módulo Base de Datos de Históricos** [Report] / auth. RECHE I. L. A. and I. A. D. L. H. DÍAZ. - 2009.
8. **Integrando mediante patrones de software una estrategia Fuzzy – Logic, en un servicio de balanceo dinámico de carga bajo CORBA** [Report] / auth. MEDINA G., JAVIER, F., MUÑOZ, J., Y MARTÍNEZ, J. - 2008.
9. **La lógica borrosa y sus aplicaciones** [Report].
10. **Màquinas de Estados Finitos** [Report] / auth. Gutiérrez Orozco Jorge Alejandro. - 2008.
11. **MODELO DE TOMA DE DECISIONES Y APRENDIZAJE EN SISTEMAS MULTI-AGENTE** [Report] / auth. BETRON J. I. G. - 1999.
12. **Sistemas Expertos Basados en Reglas** [Report] / auth. Gutiérrez José Manuel.
13. **Toma de decisiones mediante técnicas de razonamiento incierto** [Report] / auth. Reyes Saldaña José Fernando and García Flores Rodolfo.
14. **14.** Johnson, Michael K. http://www.insflug.org/COMOs/InfoSheet-Como/InfoSheet-Como.html. http://www.insflug.org/COMOs/InfoSheet-Como/InfoSheet-Como.html . [En línea] 19 de 12 de 2006. [Citado el: 28 de enero de 2010.]

http://www.wikilearning.com/tutorial/caracteristicas_principales_de_linux caracteristicas_de_linux/20536-2.

15. Luis Giraldo, Yuliana Zapata. HERRAMIENTAS DE DESARROLLO DE INGENIERIA DE SW PARA LINUX. s.l. : Monitoria de Ingesoft, 2005.

16. Conferencia 1 "Introducción a la Ingeniería de software". **autores, colectivo de.** pagina 9 : s.n., curso 2007- 2008.

17. **RAZONAMIENTO CON INCERTIDUMBRE, Ingeniería Superior Informática** [Report]. - Curso 20062007

18. **An Open-Source Fuzzy Logic Library** [Book Section] / auth. Zamzinski Michael // AI GAME PROGRAMMING WISDOM.

19. **Análisis del proyecto** [Book Section] // Desconocido / book auth. Desconocido.

20**. Control** [Book Section] / auth. Juárez Herrero José M., Cañadas Martínez José J. and Marín Morales Roque.

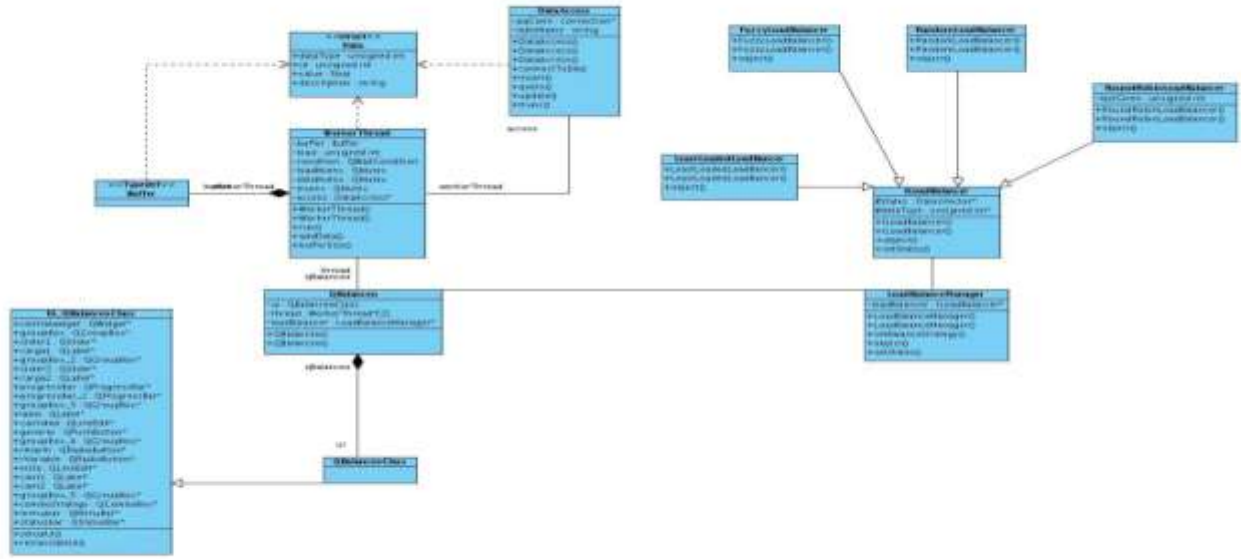
21**. Estudio comparativo de sistemas de inferencia y métodos de defuzzificación aplicados al control difuso del péndulo invertido.** [Report] / auth. Cárdenas E. [et al.]. - Granada : [s.n.], 1993.

22**. GESTIÓN DINÁMICA DE PROYECTOS MEDIANTE LÓGICA FUZZY** [Report] / auth. MENDIBURU DÍAZ HENRY ANTONIO.

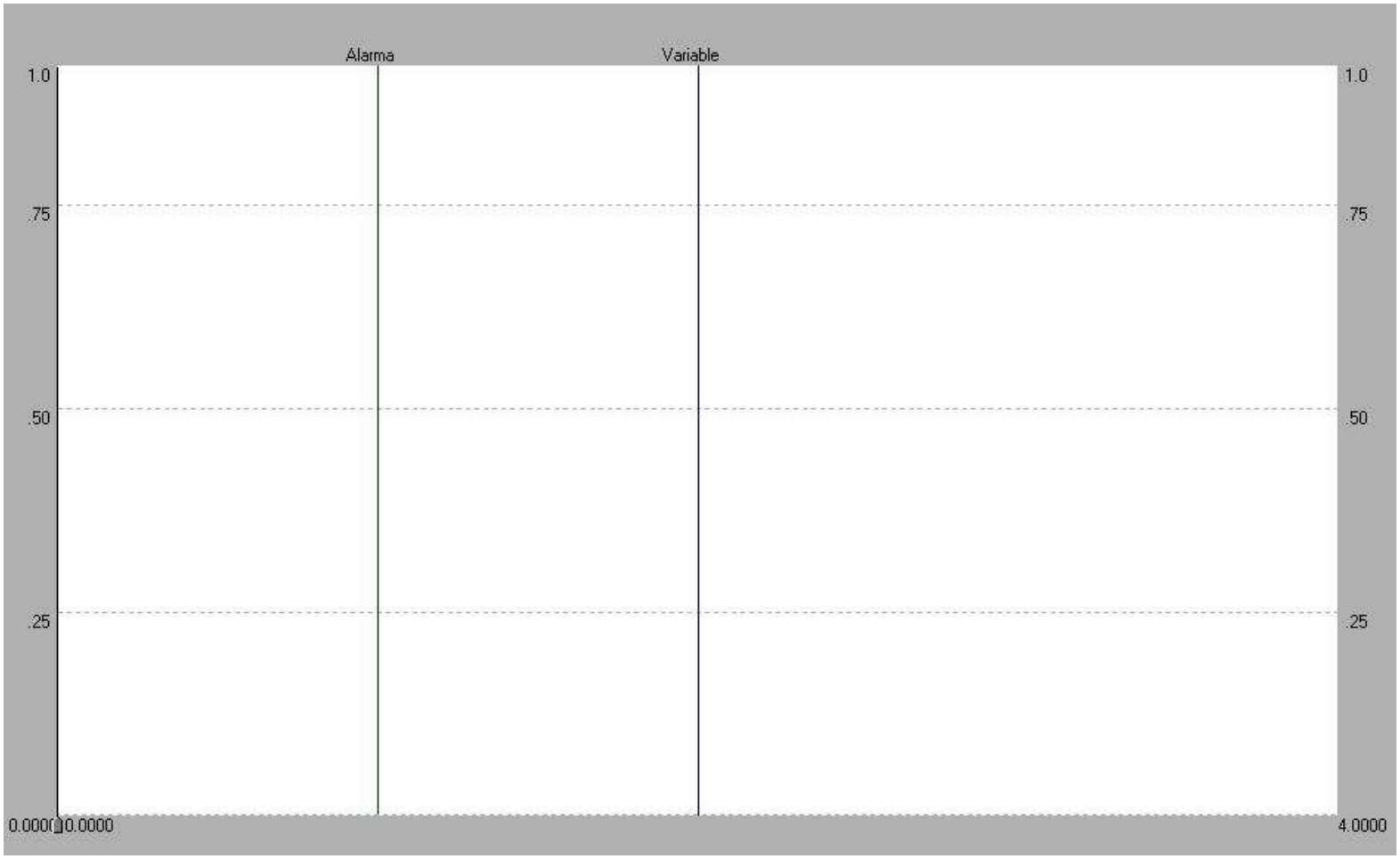
23**. NUEVO MÉTODO PARA REALIZAR EL PROCESO DE DEFUZZIFICACIÓN EN CONTROLADORES DIFUSOS.** [Report] / auth. Escamilla P. J.. - México : [s.n.].

ANEXOS

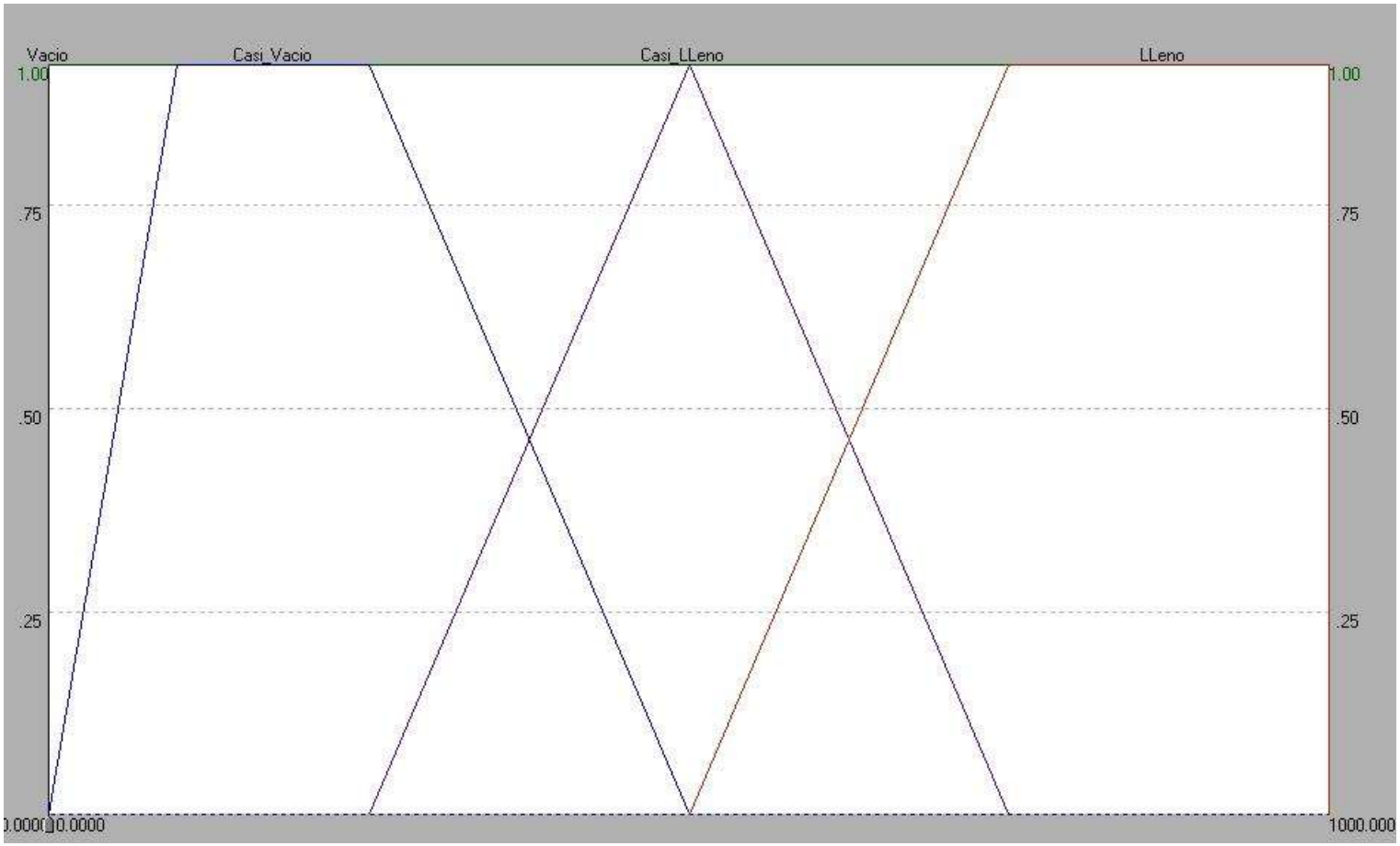
Anexo 1



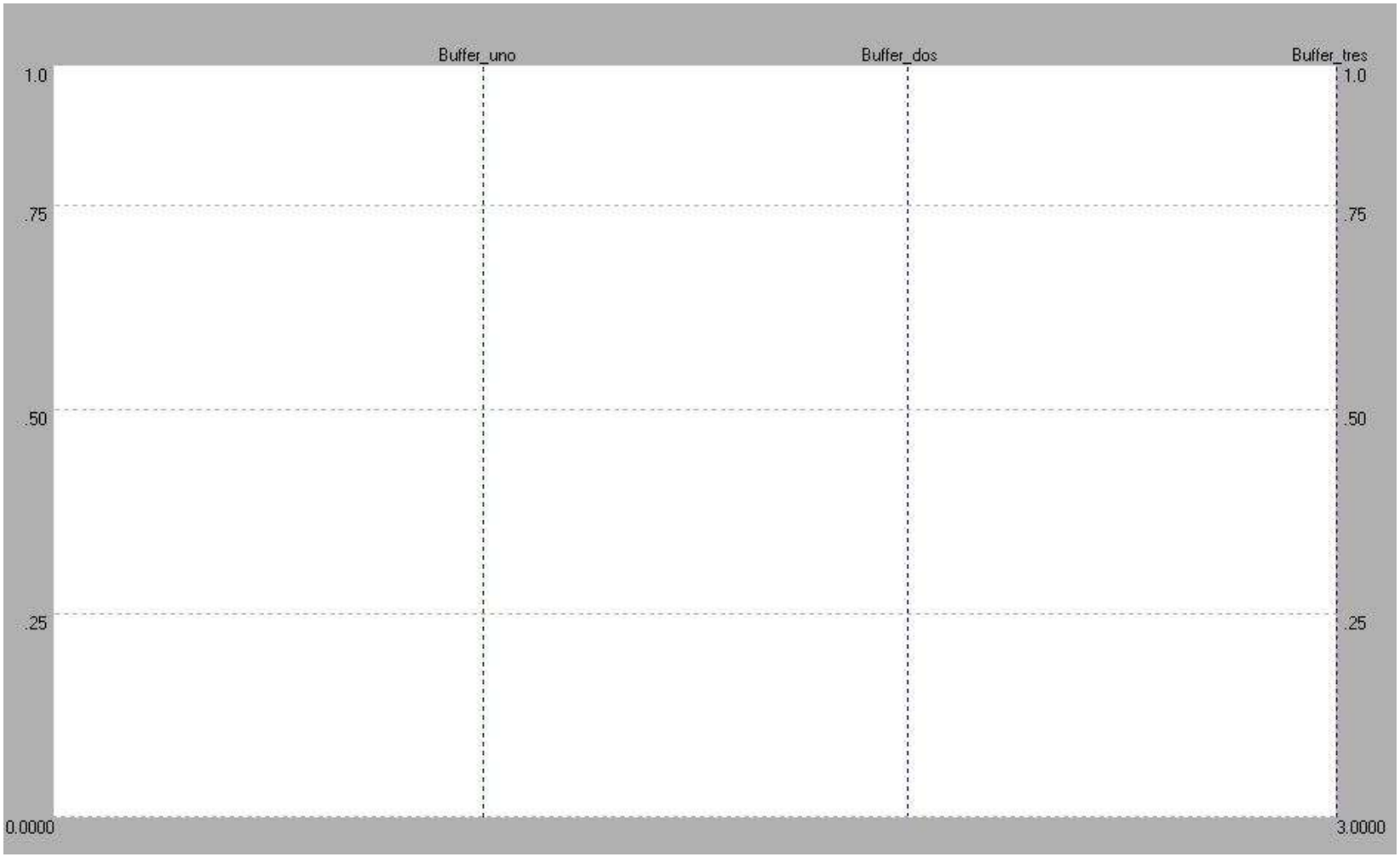
Anexo 2



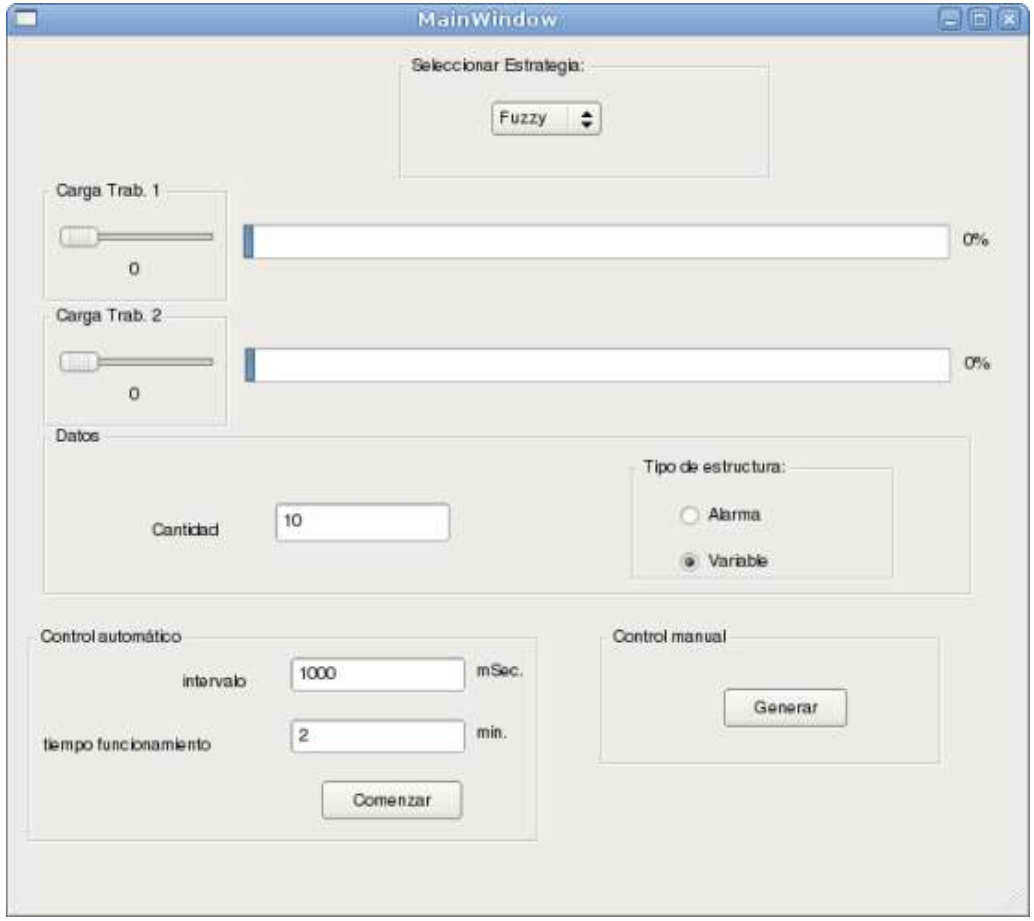
Anexo 3



Anexo 4


Buffer_uno
Buffer_dos
Buffer_tres
1.0
.75
.50
.25
0.0000
1.0
.75
.50
.25
3.0000

Anexo 5


MainWindow
Seleccionar Estrategia:
Fuzzy
Carga Trab. 1
0
0%
Carga Trab. 2
0
0%
Datos
Cantidad
10
Tipo de estructura:
Alarma
Variable
Control automático
intervalo
1000
mSec.
tiempo funcionamiento
2
min.
Comenzar
Control manual
Generar

Glosario de Abreviaturas.

AGDH: Almacenamiento y Gestión de Datos Históricos.

AUP: Agile Unified Process (Proceso Ágil Unificado).

BD: Base de Datos.

CASE: Computer Aided Software Engineering.

COA: Centro de Áreas.

COG: Centro de Gravedad.

COGS: Centro de Gravedad para Singletons.

DBN: Redes Bayesianas Dinámicas.

DBMS: Sistemas de Gestión de Bases de Datos.

FCL: Fuzzy Control Language (Lenguaje Control Difuso).

FFLL: Free Fuzzy Logic Library.

FSM: Finite State Machines (Máquinas de Estado Finito).

GUI: Interfaz Gráfica de Usuario.

HMM: Modelos Ocultos de Markov.

IEC: International Electrotechnical Commission.

JAD: Joint Application Development.

JRE: Java Runtime Environment.

LL: Least Loaded.

POO: Programación Orientada a Objetos.

RUP: Rational Unified Process (Proceso Racional Unificado).

RAD: Desarrollo Rápido de Aplicaciones.

RM: Máximo por la Derecha.

SCADA: Sistema de Control y Adquisición de Datos.

UCI: Universidad de las Ciencias Informáticas.

UML: Unified Modeling Language (Lenguaje Unificado de Modelado).

XP: Extreme programming (Programación extrema).

Glosario de Términos.

Balance de carga: La técnica usada para compartir el trabajo a realizar entre varios procesos, ordenadores, discos u otros recursos, es el proceso de compartir los recursos computacionales.

Defuzzificación: Dado los valores de pertenencia M (y) de todos los conjuntos que forman parte del universo de discurso de una variable se obtiene su valor numérico.

Fuzzificación: Para cada variable numérica de entrada se obtiene su valor de pertenencia a cada uno de los conjuntos difusos en que se divide el universo de discurso.

International Electrotechnical Commission: La Comisión Electrotécnica Internacional (CEI o IEC, por sus siglas del idioma inglés International Electrotechnical Commission) es una organización de normalización en los campos eléctrico, electrónico y tecnologías relacionadas. Desarrollan Numerosas normas conjuntamente con la ISO (normas ISO/IEC).

Módulo de AGDH: Módulo del SCADA Nacional que procesa y almacena los datos, como datos históricos.

Reglas: Son los componentes de if-then del sistema difuso. También se conoce colectivamente como una matriz asociativa borrosa (FAM).

SCADA: SCADA proviene de las siglas de Supervisory Control And Data Adquisition. Traducido al español: Adquisición de Datos y Control Supervisor. Utilizan la computadora y las tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales.