UNIVERSIDAD DE LOS ANDES

CONSEJO DE ESTUDIOS DE POSTGRADO

DOCTORADO EN CIENCIAS APLICADAS

Arquitectura Inteligente de Desarrollo de Software

Trabajo presentado como requisito final para optar al título de Doctor en Ciencias Aplicadas

Autor:

Ing. M.Sc. Blanca T. Abraham Zoghbi

Tutor:

Dr. Jose L Aguilar Castro

Mérida, Junio 2010

Dedicatoria

Cuando se estudian áreas como la inteligencia artificial, arquitecturas de sistemas y reflexión, nos damos cuenta de lo complejo que es el ser humano como sistema compuesto por múltiples mecanismos de coordinación y componentes que se integran para formarlo.

Este trabajo está dedicado a Dios; no solo por crear los mejores sistemas que conocemos, sino por darme el privilegio y el honor de tener el mejor esposo (Karim); los mejores hijos (Karina y Abraham); los mejores hermanos (Nany, Andrés y Eliche); los mejores padres (Nancy y Nagib), un sobrino hermoso (Samuel) y por supuesto al mejor tutor quien es parte fundamental de esta tesis (Jose Aguilar).

Agradecimientos

Al Prof. Leiss por su apoyo y asesoría en el desarrollo de este trabajo.

Al programa ECOS-Nord n° V04M01, por brindarme la oportunidad de realizar parte de este trabajo en INRIA Francia.

Al CDCHT-ULA proyecto no. I-821-05-02-ED Título: ARQUITECTURA INTELIGENTE DE DESARROLLO DE SOFTWARE por su aporte financiero para la realización de este trabajo.

A todo el equipo del INRIA por su asesoramiento.

A Tania Jimenez y Eitan Altman por recibirme en su casa y ser como mi familia en Francia.

A Bill Jorgensen por el apoyo que me bridó para la presentación de esta tesis.

A mis amigos de Fundacite Mérida, Cenditel, Cemex y PBS por brindarme su apoyo permanente durante todo este tiempo.

Tabla de Contenido

Índice de Figuras:

Índice de Tablas:

Resumen

Esta tesis doctoral propone dos algoritmos inteligentes de selección de componentes de software los cuales se han diseñado utilizando inteligencia artificial colectiva; y que se integran con una arquitectura inteligente de desarrollo de software que se implementa como un middlware reflexivo a través del cual se hace monitoreo de una aplicación basada en componentes permitiendo tomar decisiones sobre los cambios necesarios para mantener dicha aplicación funcionando bajo ciertas condiciones requeridas.

Se definieron e implementaron casos de uso para probar los algoritmos de selección y el middleware reflexivo, dichos casos están orientados a mantener niveles de seguridad y rendimiento de las aplicaciones utilizadas. De los resultados obtenidos en los experimentos realizados se demuestra la efectividad de los algoritmos de selección, del middleware reflexivo y de cómo interactúan juntos para el mantenimiento, monitoreo y desarrollo de software basado en componentes.

Capítulo 1

Aspectos Introductorios.

1.1. Motivación:

En el presente trabajo se define y se modela una arquitectura inteligente para el desarrollo de software, permitiendo la integración de componentes de software reutilizables en el desarrollo de aplicaciones. En los últimos tiempos, el desarrollo de software está más orientado a la integración y configuración de componentes que a la construcción total de los sistemas [36,41,65]. Esto se debe a diferentes razones, entre ellas, el hecho de que los componentes existentes ya están probados, tienen soporte, y por ende, ahorran tiempo a los proyectos de desarrollo de sistemas. Sin embargo, algunas de las desventajas de estos componentes son que se definen en su mayoría como cajas negras, como también que en ocasiones pueden incorporar a los sistemas fallas de seguridad y fallas de desempeño.

Con la finalidad de mejorar la calidad del software basado en componentes, en este trabajo se plantea utilizar ideas provenientes de diferentes campos de la computación, como lo son la computación reflexiva, la teoría de agentes, la arquitectura de software, y la ingeniería de componentes. La incorporación de la computación reflexiva nos permite monitorear y hacer cambios a los sistemas en tiempo de ejecución. La teoría de agentes, y particularmente la inteligencia artificial distribuida, nos permiten desarrollar modelos distribuidos para la resolución de los problemas planteados en este trabajo, basados en conceptos de autonomía, auto-organización, entre otros. En específico, esas ideas son usadas en el problema de búsqueda y selección de componentes, y en el diseño de un middleware reflexivo. El concepto de arquitectura de software nos permite caracterizar la estructura global del sistema, y el funcionamiento y las interacciones entre sus componentes. Esto facilita, tanto la caracterización de los procesos que compondrán al middleware reflexivo propuesto en la tesis, el cual está orientado a darle soporte a aplicaciones basadas en componentes, como también, la identificación de los agentes para la implementación de los algoritmos de búsqueda y del middleware que se proponen en esta tesis.

En particular, la integración de las áreas antes mencionadas nos ha permitido proponer dos algoritmos de búsqueda y un middleware reflexivo, el cual hace uso de dichos algoritmos. El middleware reflexivo

manipula en tiempo de ejecución aplicaciones de software basadas en componentes, a fin de incrementar la seguridad y mejorar el rendimiento de las mismas, entre otras cosas.

De esta manera, en la tesis se plantea que una arquitectura inteligente de software puede facilitar la construcción, el soporte y el monitoreo, en tiempo de ejecución, de aplicaciones de software basadas en componentes. Dicha arquitectura requiere de algoritmos de selección y búsqueda de componentes de software, y del uso de un middleware reflexivo que monitoree la aplicación basada en componentes, e induzca cambios en ella si los objetivos para los que se configuró la misma son violentados.

1.2. Objetivo General:

Este trabajo tiene como objetivo diseñar, desarrollar e implementar una arquitectura inteligente de software que facilite la construcción, el soporte y el monitoreo en tiempo de ejecución, de aplicaciones de software basadas en componentes.

1.3. Objetivos Específicos:

- Identificar la utilización de la Inteligencia Artificial Distribuida en la integración de componentes de software para la construcción de aplicaciones.
- Profundizar sobre los aspectos teóricos relacionados a la computación reflexiva, la inteligencia artificial distribuida, la arquitectura de software, la ingeniería del software (aspectos, dominio, componentes), y el procesamiento distribuido
- Diseñar un modelo de referencia de una Arquitectura Inteligente de Software.
- Proponer un modelo para la integración de componentes reutilizables de software en tiempo de ejecución.
- Desarrollar algoritmos de búsquedas de componentes de software en Internet, basados en la teoría de agentes

1.4. Estado del Arte:

Durante los últimos años se han desarrollado importantes esfuerzos por incorporar diferentes teorías a la implementación, diseño y administración de sistemas computacionales. El uso de inteligencia artificial colectiva, computación autonómica y reflexión, es cada vez más frecuente cuando se desarrollan herramientas computacionales, para mejorar aspectos de vital importancia como rendimiento y seguridad.

Esta tesis se enfoca en el desarrollo y gestión de sistemas computacionales considerando dos aspectos: la búsqueda y selección automática de los componentes de software que integrarán el sistema, y un middleware que monitorea, evalúa y administra dicho sistema en tiempo ejecución.

En el área de algoritmos de búsqueda de componentes de software, los trabajos más relevantes se resumen a continuación.

Agora es un prototipo desarrollado por el Software Engineering Institute, de la Universidad de Carnegie Mellon [67]. El objetivo de Agora es crear una base de datos indizada automáticamente, que clasifica los componentes de software por tipo, como por ejemplo si el componente es de tipo JavaBeans o es un ActiveX Control, etc. Agora combina introspección con motores de búsqueda para reducir los costos involucrados en la selección y búsqueda de componentes. La introspección permite identificar las interfaces de los componentes, a partir de la cual se hace la clasificación de los mismos para una búsqueda más eficiente. Sin embargo, aun cuando la introspección es importante y necesaria, no es suficiente para hacer búsquedas efectivas de componentes en Internet.

En [23] se presenta un repositorio con la descripción de componentes de software que son públicos en la web, los cuales pueden ser reusados en diferentes desarrollos de sistemas. En esta arquitectura, los componentes de software son descritos utilizando archivos XML. Los aspectos que se describen de un componente son el lenguaje de programación que utiliza, el nombre del programador o grupo que lo desarrolla, las dependencias del mismo con otros componentes, sistema operativo donde puede ser ejecutado, la fecha de creación, nombre y versión, entre otras cosas. Este proyecto desarrolló un sitio web que permite hacer la búsqueda de componentes manualmente, según las necesidades del usuario, creando enlaces a las páginas web de cada uno de los componentes que están descritos en el

repositorio.

En [10], una nueva metodología para clasificación y obtención de componentes de software, basada en requerimientos de usuarios, es desarrollada. El esquema de clasificación que se utiliza es un algoritmo genético, el cual clasifica a los componentes en grupos (clusters). Cuando un usuario desea buscar y seleccionar un componente, este verifica las características deseadas, y las mismas se comparan con los clasificadores de los grupos. El grupo con mayor número de características similares a las requeridas, es el que se le presenta al usuario. Algunas de las características utilizadas por esta metodología son: la plataforma de implementación requerida por el componente, la funcionalidad, el lenguaje en el que está desarrollado, el sistema operativo, el precio, el uso de memoria, el tipo encriptamiento de datos, entre otras cosas. El sistema construye un string binario que identifica si el componente posee o no cierta característica, esto permite catalogarlo y agruparlo según su tipo.

Odyssey-Search [16] es un sistema multiagente que busca componentes en la web según las preferencias de los usuarios, y el dominio de la aplicación. Odyssey utiliza básicamente dos agentes, uno para filtrar los resultados según el perfil del usuario, y otro para acceder a los diferentes repositorios de componentes.

En [68] se describe un servicio web para ubicar componentes utilizando el concepto de Ecosistema de Componentes. La idea detrás de este concepto es la siguiente: cada vez que un componente utiliza un sub-componente se crea una conexión entre ellos, y mientras más usado sea un componente en particular mayor cantidad de conexiones tendrá. Así; mientras mayor número de conexiones posea un componente, el mismo estará dentro de los primeros en la lista que aparece al hacer una búsqueda.

Existen muchas otras investigaciones en el área de motores de búsqueda, no solo para componentes de software, sino para recursos, documentos, páginas y sitios web, entre otros. Esta tesis presenta un algoritmo de selección de componentes basado en inteligencia artificial colectiva. Cada componente es tratado como una unidad, los requerimientos de selección son comparados con cada grupo de características que cada componente posee, usando archivos XML y no string binarios, como en el caso de [10]. Este algoritmo puede ser implementado como un componente más de cualquier sistema [2,6], como es el caso en el middleware reflexivo desarrollado en esta tesis doctoral [1]. Las ventajas que brinda el algoritmo son: puede ser utilizado con cualquier repositorio de componentes; permite el uso

de archivos XML no solo para describir y evaluar componentes, sino también para configurar de manera dinámica el algoritmo (por ejemplo, si no se están obteniendo los resultados esperados, se pueden incrementar el número de agentes a utilizar, o se pueden modificar los requerimientos de búsqueda); y está basado en un modelo que no solo busca componentes de software, sino además "sugiere" aquellos que tengan mayor probabilidad de funcionar de una manera eficiente al ser integrado a una aplicación en particular.

La segunda parte de esta tesis está basada en un middleware reflexivo. Los trabajos más relevantes en esta área son los siguientes: En [24] se propone una arquitectura inteligente para la asignación automática de recursos en clusters de computadoras. Esta arquitectura no es costosa, y puede ser implementada en hardware y software existentes, es escalable, recicla y asigna recursos con mínimas interrupciones, posee una estructura modular que facilita la incorporación de nuevas funcionalidades. La arquitectura está compuesta por agentes que monitorean sistemas y servicios, corrigiendo fallas en tiempo de ejecución cuando esto es posible. Los agentes se ejecutan cada 5 minutos para buscar fallas de CPU, memoria, disco duro y red. Esta arquitectura es similar a la propuesta en esta tesis para el caso de estudio de rendimiento, sin embargo, la ventaja de nuestra propuesta es que posibilita reemplazar componentes de software a fin de mejorar el rendimiento de la aplicación bajo monitoreo.

GridKit Middleware [18,25] es otro middleware configurable desarrollado usando el modelo de componentes OpenCom. Este trabajo está orientado al manejo de recursos en Grids Parte de las funcionalidades más importantes de GridKit son la capacidad de descubrir recursos (nodos), manejarlos, y mantener la seguridad del Grid, específicamente en cuanto a autenticación y confidencialidad. A diferencia de GridKit, nuestro middleware está orientado a la búsqueda, reemplazo y configuración de componentes de software, a fin de mejorar el rendimiento y la seguridad de las aplicaciones.

DynamicTAO [47] fue desarrollado en la Universidad de Illinois, con el objetivo de manejar aplicaciones en tiempo real. Sin embargo, puede ser utilizado para cualquier otro sistema que este supuesto a ser ejecutado durante un largo periodo de tiempo. DynamicTAO utiliza técnicas de reflexión para adaptarse al entorno de ejecución, específicamente al ancho de banda, a la carga de la CPU, y a la disponibilidad de memoria, características que deben conocerse a priori. Este middleware está orientado a aplicaciones distribuidas, y se auto-organiza cargando componentes que no son parte del middleware base para

manipular los recursos, sin cambiar la aplicación como tal. Actualmente este proyecto no está activo.

Por otro lado, no existen muchos trabajos que integren técnicas de computación reflexiva, inteligencia artificial distribuida y componentes de software. Una de las investigaciones más parecidas que se encontró fue el trabajo *"Intelligence engine software architecture",* el cual es una patente registrada el 28 de julio del 2009, y consiste en tres capas: una que sirve como sensor y recibe toda la información de diferentes aplicaciones; la capa de conocimiento que identifica patrones en la información que recoge el sensor, y la guarda a fin de poder predecir comportamientos y patrones de las aplicaciones; y por último, una capa inteligente que genera reportes y eventos en base a los patrones reconocidos. Esta arquitectura está diseñada para predecir posibles fallas y comportamientos, y no para la corrección de las mismas, o cambio de los sistemas en tiempo ejecución, como si lo hace nuestra arquitectura [1].

Existen diferentes trabajos relacionados con middlewares adaptativos para arquitecturas basadas en sistemas CORBA [34,35,49,51,56], sin embargo, solo se mencionaran los que tienen ciertas similitudes con los trabajos descritos en esta tesis. Por ejemplo, en [34] se presenta un middleware para tolerancia a fallas de sistemas en tiempo real. Este provee capacidades de predicción del comportamiento del sistema, está diseñado específicamente para componentes CORBA, y se basa en la posibilidad de mantener redundancia en dichos componentes a fin de reemplazarlos si ocurre una falla.

One.world representa un middleware orientado a manejar tres requerimientos fundamentales [38]: portabilidad entre dispositivos tales como PDAs, celulares etc. (es decir, puede ser utilizado en diferentes dispositivos móviles); interacción entre diferentes dispositivos, e intercambio de información entre diferentes usuarios de diferentes dispositivos. One.world trabaja como una capa situada encima de sistemas operativos tradicionales, tales como Windows o Linux. One.world propone un modelo para manejar datos e información que dichos dispositivos móviles intercambian, eventos asíncronos, notificaciones de cambios entre aplicaciones, y procesos de almacenamiento de información. Este sistema ha sido probado en diferentes ambientes, y maneja la seguridad relacionada con la autenticación de usuarios. Sin embargo, en el área de políticas de seguridad todavía existen problemas que solventar, tales como la autenticación de dispositivos.

GaiaOS [63] es un meta sistema operativo basado en componentes, también llamado sistema operativo–middleware, que corre por encima de sistemas operativos convencionales. La infraestructura

que utiliza GaiaOS para la integración de diferentes plataformas esta basada en CORBA. GaiaOS esta compuesto de dos unidades, la primera es un bus de objetos unificados que permite manipular componentes heterogéneos que corren en el sistema, tales como componentes CORBA, archivos nativos del sistema, y scripts. En esta unidad residen los componentes abstractos propios de GaiaOS. La otra unidad es el kernel, donde residen los servicios esenciales que implementan funcionalidades como descubrimiento de entidades, repositorio de componentes, etc.

Existen muchos otros trabajos relacionados con middleware, sin embargo solo se han mencionado algunos que tienen ciertas similitudes con el trabajo descrito en esta tesis. Básicamente, nuestro trabajo propone un middleware reflexivo que integra un algoritmo inteligente de búsqueda y selección de componentes de software, y utiliza la teoría de computación reflexiva e inteligencia artificial distribuida para manipular en tiempo de ejecución sistemas basados en componentes. Las ventajas de nuestra propuesta son la facilidad de configuración y manipulación a través del uso de archivos XML, que pueden ser editados para modificar la configuración inicial en cualquier momento de la ejecución del sistema; así mismo, y a diferencia de propuestas como las descritas en [18,24,25,47], el middleware está diseñado para resolver problemas a través de la modificación y manipulación del software en el sistema base; por otro lado, este modelo de middleware es suficientemente flexible para ser implementado en diferentes tipos de arquitecturas, tales como en Grids o Clusters; y finalmente, el mismo integra un innovador algoritmo de búsqueda de componentes fácilmente configurable, que potencia la capacidad de adaptación y eficiencia de los sistemas que lo utilicen.

1.5. Organización de la Tesis:

Esta tesis está organizada de la siguiente manera: el capítulo uno introduce el problema a estudiar y describe el estado del arte de las áreas que abarca nuestro trabajo, seguidamente, el capítulo dos describe los conceptos teóricos más relevantes que han sido utilizados para el desarrollo de esta investigación. Los siguientes tres capítulos describen los algoritmos de búsqueda que han sido desarrollados, el middleware reflexivo implementado, y los casos de estudio que fueron analizados. En el capítulo seis se exponen las conclusiones y recomendaciones de nuestro trabajo. Como parte final del esta tesis se encuentra la bibliografía utilizada y el anexo que presenta el manual del middelware reflexivo.

Capítulo 2

Marco Teórico

2.1 Ingeniería de Componentes

La ingeniería de software es una rama de la ingeniería que comprende todos los aspectos de la producción de software. El término de Ingeniería de Software se comenzó a usar a finales de los sesenta y nace con la finalidad de formalizar los métodos de desarrollo de software que hasta ese momento estaban enfocados en general a la electrónica [59]. Así, la Ingeniería de Componentes (IC), también conocida como Ingeniería de Software Basada en Componentes (ISBC), surgió en la década de los 90s como un enfoque basado en la reutilización de componentes para el desarrollo de sistemas de software [59].

La palabra componer proviene del latín componere, cum "junto", y ponere "poner". Las partes que se ponen juntos son, etimológicamente hablando, componentes de un sistema superior, o sub-sistemas o componentes integrados en un sistema compuesto. Por definición, todos los sistemas de software contienen de una u otra forma componentes. Estos componentes resultan de la descomposición de los problemas, el cual es un método muy conocido de resolución de problemas [12].

La Teoría de componentes consiste en diseñar, construir y documentar partes de software que puedan ser probadas y reutilizadas como unidades independientes, de manera tal que su integración a sistemas más amplios sea posible. Los componentes de software se construyen mediante lenguajes de programación que tienen una gramática de integración definida explícitamente por reglas bien formadas semántica y sintácticamente propias de dichos lenguajes, lo cual permite la construcción de interfaces para el futuro uso de los mismos de una manera eficiente[20, 73]. De esta manera, los componentes encapsulan su implementación e interactúan con otros componentes a través de interfaces bien definidas.

Bajo el enfoque ISBC los desarrolladores diseñan el sistema, buscan componentes disponibles, y adaptan el diseño de sus aplicaciones a los componentes encontrados [20, 37]. El desarrollo de software a través de componentes requiere definir características funcionales (como por ejemplo, qué interfaces

utiliza y para qué, qué resultados genera etc.), y no funcionales (como el nombre del componente, la plataforma en la que ha sido desarrollado, fecha de desarrollo entre otros), a fin de seleccionar los componentes apropiados que se integraran en una aplicación.

Existen en la literatura numerosas definiciones de componentes de software, entre las cuales se pueden citar:

- Los componentes de software son unidades binarias de producción independiente que interactúan para formar un sistema funcional [1]
- “Un componente es una unidad de composición con interfaces contractuales especificadas y dependencias de contexto explícitas" [22].
- “Un componente es una parte no trivial, casi independiente, y reemplazable de un sistema que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. Un componente se conforma y provee la realización física por medio de un conjunto de interfaces” [17].
- “Un componente es una implementación de software que puede ser ejecutada sobre un dispositivo lógico o físico. Un componente implementa una o más interfaces que deben obedecer ciertas reglas, de manera que componentes desarrollados independientemente puedan interactuar de manera predecible” [12].

Un componente no es un objeto. Y al hablar de objetos, vale la pena distinguir aquí los objetos de las clases. Una clase es una definición de propiedades y funcionalidades a ser provistas por los objetos. A partir de una clase es posible instanciar objetos. Los componentes pueden contener una o más clases, y serán los usuarios de los componentes quienes soliciten la creación de las instancias de estas clases. Un componente puede tomar la forma de un archivo ejecutable o una biblioteca dinámica, que usualmente cobra vida a través de objetos, pero no es este un requisito indispensable. De hecho, los primeros componentes conocidos (aunque en su momento no se los haya definido así) fueron las bibliotecas de procedimientos [22]. Sin embargo, los objetos, con sus características de encapsulación y polimorfismo, facilitan la construcción e integración de componentes.

Las principales características de un componente de software son [17,22,42]:

- Es auto-contenido[1]
- Es accesible solo a través de sus interfaces, las cuales deben estar claramente definidas.
- Tiene un conjunto de operaciones que puede realizar, ocultando detalles de su implementación.
- Sus servicios son inmutables, es decir, los servicios que ofrece un componente a través de sus interfaces no deben variar.
- La implementación física de estos servicios pueden ser modificadas, pero no deben afectar la interfaz a la que está asociado.
- Debe tener una documentación adecuada que facilite:
    - La recuperación del componente desde un repositorio,
    - La evaluación del componente,
    - La adaptación a un nuevo ambiente.
    - Su integración con otros componentes.
- Debe ser reemplazable por una nueva versión, o por otro componente que proporcione los mismos servicios.

En esencia, la ISBC se enfoca en las siguientes características [40]:

a. Descansa en la existencia de Partes de Software (llamados componentes).
b. Emplea líneas de producción en las que el producto es elaborado mediante el ensamblaje de componentes.

Todo esto, por supuesto, reduce los tiempos de desarrollo [40]. Así, la ingeniería de componentes incrementa la posibilidad de reutilización de componentes de software, y facilita la combinación y reconfiguración dinámica de componentes existentes.

[1]Un componente es auto-contenido cuando no requiere la utilización de otros componentes para cumplir su función.

Con la finalidad de ser realmente útiles en el desarrollo de sistemas y poder ser reutilizados, los componentes deben poseer información suficiente sobre sus características técnicas en la tabla de perfil del componente. El proceso de reutilización de componentes de software se describe como sigue [13,60,64]:

- Cuando se culmina de desarrollar un componente: Se define un grupo de dominios naturales de posibles áreas de aplicación de los componentes. Se establecen las propiedades de cada componente para cada subdominio. Se desarrolla la tabla de perfil asociada a cada componente, que publica las propiedades del mismo.

Cuando se está desarrollado un sistema se decide, según el diseño preliminar del sistema, sobre qué componentes utilizar. Si no hay componente que cumpla con las condiciones que el sistema requiere, se debe construir un nuevo componente, o rediseñar el sistema.

De lo dicho anteriormente, pueden identificarse varios problemas para la construcción de sistemas reutilizando componentes [14]:

- El problema de búsqueda de componentes que satisfagan los requisitos impuestos, tanto por el cliente como por la arquitectura de la aplicación.
- La evaluación de los componentes candidatos, que aborda el problema de seleccionar los más idóneos.
- El problema de adaptación y/o extensión de los componentes seleccionados, si es necesario que se ajusten a los requisitos impuestos.
- El problema de búsqueda de componentes que no poseen información sobre sus funcionalidades y características.
- Y por último, el problema de la integración, configuración e interconexión de dichos componentes, para construir la aplicación final.

Actualmente, las plataformas más utilizadas para desarrollo de software basado en componentes son .net, de Microsoft, y J2EE, de Sun Microsystem [40]. El lenguaje de programación JAVA/J2EE es prácticamente en su totalidad orientado a objetos. La plataforma JAVA provee un buen número de

componentes que contienen diferentes definiciones y funcionalidades bastante útiles a la hora de programar [75].

Microsoft .net intenta implementar un cambio radical en la manera de construir sistemas, al posibilitar la integración de componentes de software desarrollados con .COM, C, y otros lenguajes de programación. Así mismo, .net permite la coexistencia de diferentes versiones de librerías de software en un mismo ambiente [74].

2.1.1. Gramática de integración e interfaces:

Los componentes de software interactúan con otros componentes de software a través de sus interfaces. La interface de un componente de software define la manera en que el componente será utilizado, en otras palabras, define que entradas requiere el componente a fin de generar las respuestas esperadas por otros componentes; estas respuestas, a su vez, pueden ser las entradas requeridas por las interfaces de otros componentes.

Como se dijo antes, en la tecnología de componentes la interfaz constituye el elemento básico de interconectividad. Cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. Además, debe operar correctamente con independencia de los mecanismos internos que utilice para soportar sus funcionalidades. Por ejemplo, un componente encargado de ejecutar una división de números enteros tendrá dos interfaces definidas, una para recibir el dividendo y el divisor y la otra para entregar el resultado. Así, la primera interface tendrá ciertas reglas, por ejemplo, solo permitirá números, los números deben ser enteros, y el divisor debe ser diferente de cero. Por otra parte, la interface de salida tendrá otras reglas, como por ejemplo que el número generado puede ser un decimal, o cero. La última regla es que ambas interfaces no generan ni aceptan caracteres. Todas estas reglas son definidas como parte de la gramática de integración de un componente dado.

2.1.2. Algunas librerías o modelos basados en IC:

A continuación se explicará brevemente FRACTAL, el cual es un modelo de arquitectura basado en componentes, modular, extensible, y puede ser utilizado por diferentes lenguajes de programación para diseñar, implementar y configurar sistemas y aplicaciones como sistemas operativos, middleware e interfaces graficas [32, 55]. Este modelo usa el principio de "separación de aspectos", que consiste en separar el sistema en piezas de código, o entidades ejecutables, para cada aspecto o grupo de aspectos de la aplicación.

El principio de "separación de aspectos" es aplicado a cada componente de una estructura FRACTAL, el cual está compuesto internamente por dos partes, un contenedor que maneja todos los aspectos funcionales, y un controlador que maneja cero o más aspectos no funcionales, como introspección[2], configuración, etc. El contenedor puede ser, a su vez, hecho de otro u otros componentes FRACTAL, es decir, los componentes FRACTAL pueden ser anidados y compartidos unos por otros (Fig. 1).En cuanto a los puertos, son las interfaces de entrada y salida de los componentes, y finalmente, la unión es la interdependencia entre ellos cuando forman un componente compuesto. Las interfaces de introspección y configuración, provistas por los controladores, permiten a los componentes ser desplegados y configurados dinámicamente [55].

Antes de programar un componente basado en FRACTAL, este debe ser diseñado identificando los sub-componentes que lo van a integrar. El diseño y la programación de componentes son tareas independientes. Por otro lado, es posible agrupar varios, o todos los componentes diseñados, en una pieza monolítica de código; sin embargo, no es recomendable, pues se pierden las ventajas de modularidad y adaptabilidad de la programación orientada a componentes [55]

En una aplicación basada en componentes FRACTAL, algunos de estos son dinámicos, es decir, pueden ser creados o destruidos según el requerimiento, otros componentes son estáticos, es decir, su tiempo de vida es el mismo que el tiempo de vida de la aplicación misma. Los componentes dinámicos generalmente están asociados a datos, mientras que los estáticos corresponden a servicios. Después de que cada servicio es identificado, este debe ser asignado a un componente, y es recomendable usar un

[2]Introspección: Es la capacidad que tiene un programa para observar y razonar sobre su propio estado de ejecución.

solo servicio por componente, a menos que los servicios sean fuertemente dependientes, en cuyo caso deberían estar juntos en un componente [55]. Luego de que los componentes han sido identificados, es simple encontrar las dependencias entre ellos y organizarlos como componentes compuestos, si es necesario [55].

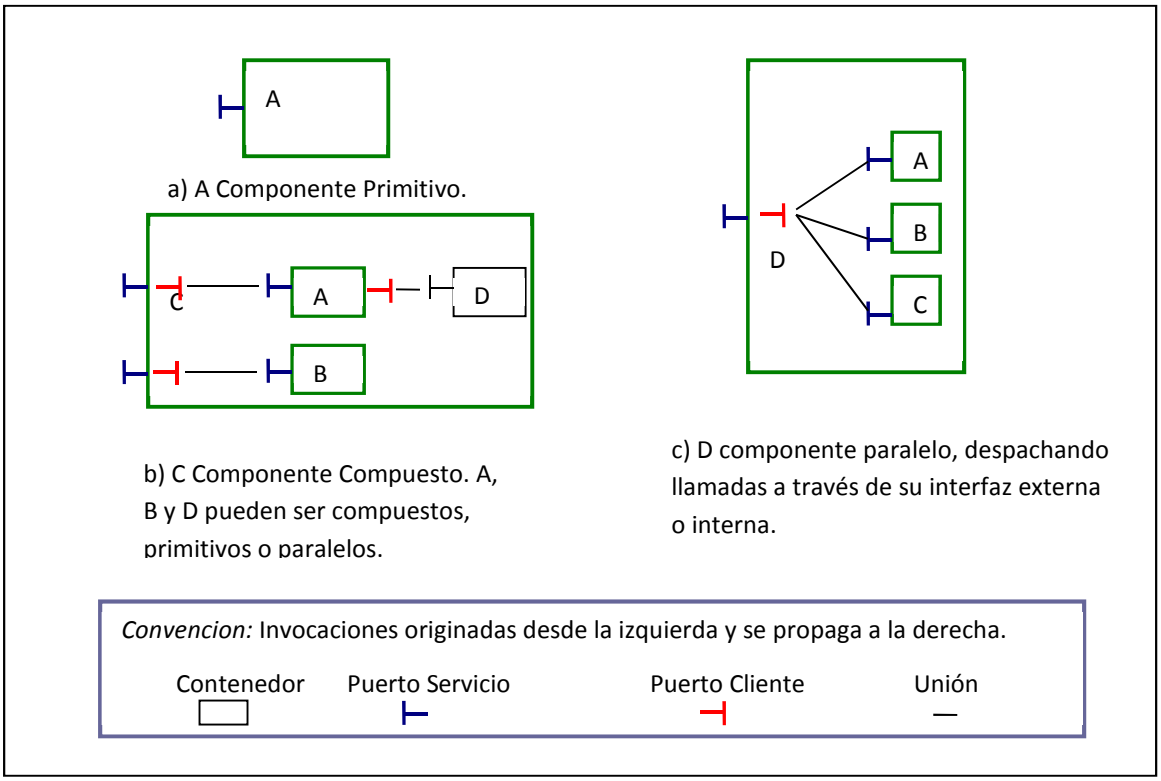

Figura 1: Arquitectura básica de componentes FRACTAL

Una de las implementaciones del modelo FRACTAL es Fractive [55], el cual combina al modelo FRACTAL con ProActive [55]. ProActive es un middleware con propiedades de computación reflexiva, que está desarrollado como una biblioteca de JAVA, e implementa funciones que permiten la programación concurrente, paralela, distribuida, y móvil. Las características más importantes de ProActive son [55]:

- Provee un patrón uniforme para la programación de objetos activos[3]
- Permite el acceso remoto a objetos, vía invocación.

[3]Los objetos activos son aquellos que poseen una hebra de ejecución y una cola de requerimientos pendientes.

- Permite la comunicación asíncrona y síncrona entre grupos de objetos.
- Permite la reflexión en tiempo de ejecución[4]
- ProActive permite encapsular:
    - Un objeto remoto accesible.
    - Una hebra con actividad asíncrona
    - Un servicio de llamadas para activar o desactivar cada objeto

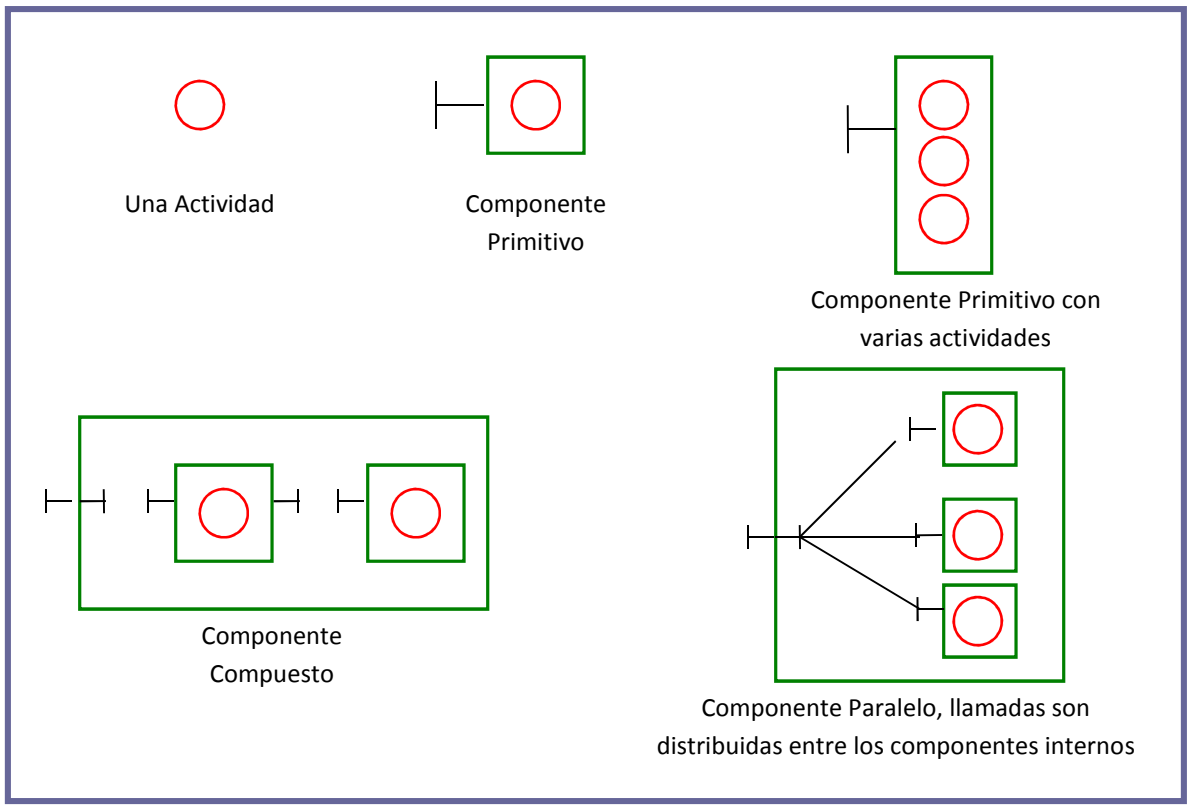

Figura 2: Diferentes clases de componentes implementados con ProActive.

[4]En ProActive la capacidad de reflexión de los objetos se hereda de las bibliotecas de reflexión de JAVA, las cuales se enfocan primordialmente a capacidades de introspección.

Fractive permite implementar el modelo Fractal utilizando ProActive como base para la creación de componentes primitivos o compuestos, consiguiendo unificar la noción de componentes y de objetos activos en una misma aplicación [55].

Como se muestra en la figura 2, un componente ProActive puede ser primitivo, cuando está definido por una o varias actividades (implementación JAVA) que proveen un servicio, a través de un puerto o interface definida para ello. También pueden ser componentes compuestos, formados por un grupo de componentes que brindan servicios. Finalmente, los componentes paralelos son aquellos que distribuyen llamadas a sus componentes internos para que se ejecuten en paralelo [33].

2.1.3. Técnicas afines a la IC

Técnicas cercanas a la Ingeniería de Componentes son la Ingeniería de Dominios y de Aspectos, las cuales pertenecen también al área de Ingeniería de Software [59].

La Ingeniería de Aspectos: Es una técnica de la ingeniería de componentes de software, la cual caracteriza las propiedades (llamadas aspectos en este caso) de los componentes (interfaces de usuarios, seguridad, distribución, etc.), con la finalidad de facilitar el diseño, la implementación y la reutilización de los mismos [39]. La programación orientada a aspectos (PA) está orientada a facilitar el diseño adecuado de las aplicaciones, basado en una mejor separación de los aspectos que caracterizan a las aplicaciones. *La Ingeniería de Dominio* consiste en identificar, construir, catalogar y diseminar un conjunto de componentes de software, que son usados específicamente en un determinado dominio de aplicación [59]. Esto permite, normalmente, la reutilización de dicho conjunto en múltiples proyectos dentro del mismo dominio de aplicación. En esencia, el análisis del dominio es similar a la ingeniería del conocimiento. Durante el análisis del dominio ocurre la extracción de características y propiedades del dominio que permitirán determinar los componentes que deberán pertenecer al mismo y sus características. Por ejemplo, si se desea programar un componente para la medición de azúcar en la sangre en tiempo real, se debe considerar el dominio donde se implementará, en este caso el médico, por lo que se requerirá determinar si será interno al cuerpo de un paciente o externo, que tipo de interfaces son necesarias programar, con que otros componentes se desea integrar, etc.

2.2. Arquitectura de Software [AS]:

Una Arquitectura de Software, también denominada Arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software. La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces, y la comunicación ente ellos. Es el resultado de ensamblar un cierto número de elementos arquitectónicos (por ejemplo, componentes de software) de forma adecuada, para satisfacer funcionalidades y requerimientos de desempeño de un sistema, así como requerimientos no funcionales como confiabilidad, escalabilidad, portabilidad, y disponibilidad. De esta manera, la arquitectura de software establece las bases para que analistas, diseñadores, programadores, etc., trabajen colaborativamente para alcanzar los objetivos y necesidades del sistema a crear.

Toda arquitectura de software debe describir diversos aspectos del software, como por ejemplo el lenguaje de programación a usar, los tipos de datos que se requieren, etc. Generalmente, cada uno de estos aspectos se describe de una manera comprensible, utilizando distintos modelos o vistas. Es importante destacar que cada uno de ellos constituye una descripción parcial de una misma arquitectura, y es deseable que exista cierto solapamiento entre ellos. Cada paradigma de desarrollo[5] exige diferentes tipos de vistas o modelos para describir una arquitectura. No obstante, existen al menos tres vistas fundamentales en cualquier arquitectura:

- La vista estática: describe qué componentes tiene la arquitectura.
- La vista funcional: describe qué hace cada componente.
- La vista dinámica: describe cómo se comportan los componentes a lo largo del tiempo, y cómo interactúan entre sí.

Las vistas o modelos de una arquitectura pueden expresarse mediante uno o varios lenguajes. El más obvio es el lenguaje natural, pero existen otros lenguajes, tales como los diagramas de estado, los diagramas de flujo de datos, etc. Estos lenguajes son apropiados únicamente para un modelo o vista.

[5]Entre los paradigmas actuales de desarrollo de software podemos citar [59] el modelo en cascada, modelo espiral, modelo basado en prototipos, basado en componentes, basado en objetos, entre otros.

Afortunadamente, existe cierto consenso en adoptar UML (Unified Modeling Language, lenguaje unificado de modelado) como lenguaje único para todos los modelos o vistas. El UML es el lenguaje gráfico para modelado de sistemas de software más conocido y utilizado actualmente, se usa para visualizar, especificar, construir y documentar un sistema de software. Algunos de los diagramas que propone el lenguaje UML son [58]:

- Los diagramas de estructura, que enfatizan los elementos que deben existir en el sistema modelado, como las clases, componentes, objetos, etc.
- Los Diagramas de comportamiento, que se enfocan en lo que debe suceder en el sistema modelado, como por ejemplo las actividades, casos de uso y estados.
- Los Diagramas de Interacción, los cuales pueden considerarse como parte de los diagramas de comportamiento. Estos muestran el flujo de control y de datos entre los elementos del sistema modelado, así como la colaboración e interacciones entre ellos, entre otras cosas.

Generalmente, se adopta una arquitectura de software conocida en función de sus ventajas e inconvenientes para cada caso en concreto. Las arquitecturas más comunes son (ver fig. 3) [61]:

- Monolítica. Donde el software se estructura en grupos funcionales muy acoplados
- Cliente-servidor. Donde el software reparte su carga de cómputo en dos partes independientes, una de ellas (servidor) realizando las funciones requeridas por el otro (cliente).
- Arquitectura de tres niveles. Especialización de la arquitectura cliente-servidor, donde el trabajo se divide en tres partes con un reparto claro de funciones: una para la presentación, en la cual se utilizan componentes relacionados a la interfaz gráfica; otra para el cálculo, donde los componentes son en su mayoría de procesamiento y cálculo; y otra para el almacenamiento, en la cual se incorporan componentes de manejo de bases de datos, archivos de texto etc. Cada parte solamente tiene relación con la siguiente.

La Arquitectura de Software se encuentra, en una etapa aún formativa [61]. Sus teóricos no se encuentran todavía en condiciones de asegurar que utilizando AS se podrá producir software dentro de un plan predecible, con una alta calidad, y un presupuesto más razonable [61].

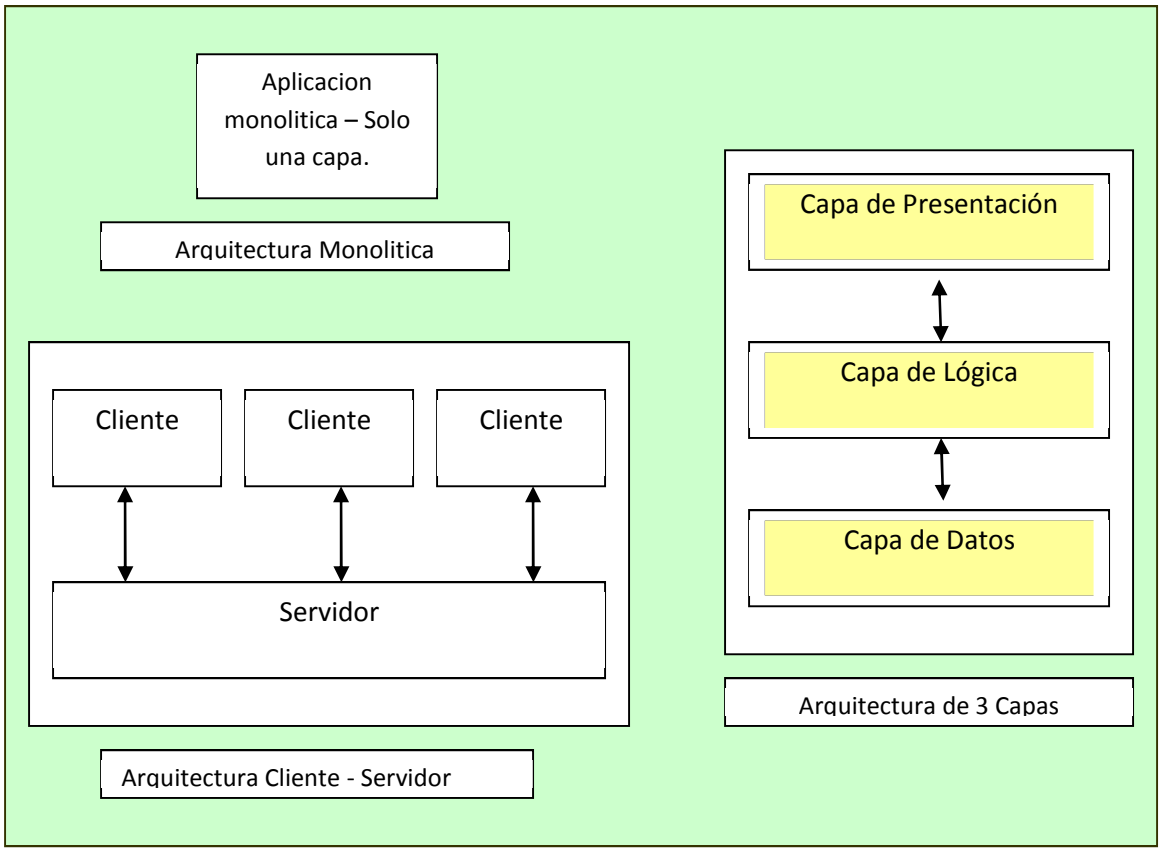

Figura 3: Ejemplos de Arquitecturas de software

2.2.1. Arquitectura de Software basada en componentes

El objetivo de la Arquitectura de Software basada en componentes es construir aplicaciones complejas, mediante ensamblado de módulos (componentes) que han sido previamente diseñados por otros desarrolladores o empresas de software a fin de ser reusados en múltiples aplicaciones [73]. La arquitectura del software de una aplicación basada en componentes consiste en uno o un número pequeño de componentes específicos a la aplicación (que se diseñan específicamente para ella), que hacen uso de muchos otros componentes prefabricados, que se ensamblan entre sí para proporcionar los servicios que se necesitan en la aplicación (ver fig. 4) [52].

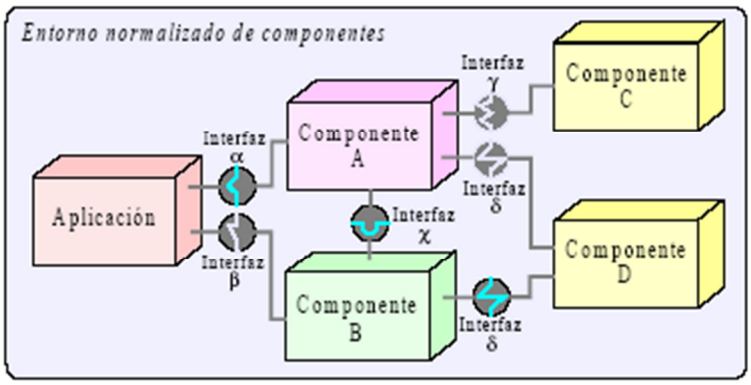

Figura 4: Arquitectura del software de una aplicación basada en componentes

Cada cubo es un componente de software, el cual posee una o más interfaces que le permiten establecer la comunicación e integración con los otros componentes de software que componen la aplicación.

2.3. Inteligencia Artificial [IA]:

Para estudiar la inteligencia sería necesario comprender cómo se adquiere, se representa y se almacena el conocimiento; cómo se genera y se aprende un comportamiento inteligente; cómo se desarrollan y se usan las motivaciones, emociones y prioridades; cómo las señales sensoriales son transformadas en símbolos; cómo se manipulan los símbolos en un contexto lógico para razonar sobre el pasado y para planificar el futuro; cómo los mecanismos de la inteligencia producen los fenómenos de la ilusión, las creencias, las esperanzas, los temores, los sueños, la bondad, el amor, entre otras cosas. Comprender estas funciones a partir de una base teórica sólida sería un logro científico de la misma escala que la física nuclear, la relatividad, y la genética molecular [54].

La Inteligencia Artificial Distribuida (IAD) se puede ver como un subcampo de la IA que busca la solución colaborativa de problemas mediante un grupo distribuido de entidades [66]. Uno de los conceptos más importantes en el área de IA es el de “Agentes”.

Un agente es una entidad física o abstracta que puede percibir su ambiente a través de sensores, es capaz de evaluar tales percepciones, tomar decisiones por medio de mecanismos de razonamientos

sencillos o complejos, comunicarse con otros agentes para obtener información, y actuar sobre el medio en el que se desenvuelve a través de ejecutores [66]. Por otro lado, un agente inteligente es una entidad de software que tiene conocimiento sobre un problema en particular, sobre su entorno, y que podría interactuar con otros agentes si fuese necesario, realizando de manera autónoma e independiente tareas que se consideran inteligentes, como razonar sobre un problema, programar actividades, entre otras cosas [66, 75].

La IAD se centra en la resolución de problemas mediante la aplicación, tanto de técnicas de la IA como de los Sistemas Distribuidos. En general, los sistemas basados en la IAD son compuestos y cooperativos, e integran al menos dos agentes, los cuales interactúan para la resolución de un problema dado. Los agentes son frecuentemente heterogéneos, y deben tener cierto grado de autonomía [21]. Los agentes deben ser capaces de interactuar, negociar, cooperar, y hasta competir con otros para llevar a cabo sus tareas, las cuales pueden ser individuales o grupales [21,54,66,75].

Los agentes de software pueden tener diferentes comportamientos:

- Comportamiento Proactivo: El agente no actúa en respuesta del ambiente, sino que es capaz de tomar la iniciativa y efectuar tareas, es individualmente inteligente, y se guía por sus metas y aspiraciones.

- Comportamiento Reactivo: El agente actúa en base a los cambios que se generan en su ambiente. Los agentes reactivos no son individualmente inteligentes, sus acciones se basan exclusivamente en su entorno. Los sistemas reactivos, por lo general, están compuestos por un gran número de agentes reactivos que realizan acciones colaborativamente. Para esto, es necesario tener en cuenta nuevas teorías de cooperación y comunicación que permitan la integración de estas acciones.

- Comportamiento Social: Es la capacidad de los agentes de interactuar con otros agentes, o con humanos, cuando se requiere.

Así mismo, generalmente los agentes deben poseer autonomía, siendo capaces de interactuar sin intervención directa, mantener control de su estado interno y sobre sus propias acciones, y modificar, si es necesario, su comportamiento. Otra característica importante de los agentes es la movilidad, que es

la habilidad del agente de moverse de ambiente, o de nodos en una red [62,66]. En general, un agente de software se caracteriza como un proceso informático al que se le pueden atribuir, además de las características antes mencionadas, algunas de las siguientes características [62]:

- Razonamiento: El razonamiento puede ser inductivo o deductivo, el inductivo consiste en obtener conclusiones generales a partir de datos particulares. Por ejemplo, de la observación repetida de objetos o patrones se establecen conclusiones. El Razonamiento Deductivo es un proceso de razonamiento en el que se extraen conclusiones acerca de una proposición.
- Racionalidad: Los agentes tienen un conjunto de objetivos predefinidos, y emprenden acciones para conseguirlos. La decisión de cual acción seguir y en qué momento hacerlo es definida según un principio de racionalidad, es decir, prefieren la acción más prometedora o eficiente para conseguir sus metas.

Dos ramas de la IAD son los sistemas multiagente, los cuales se enfocan primordialmente en la coordinación de los agentes, y los sistemas de resolución distribuida de problemas, los cuales se enfocan en la descomposición de los problemas y síntesis de las soluciones propuestas para cada subproblema en que fue descompuesto el problema.

2.3.1. Sistemas Multiagentes:

Desde el punto de vista de la IAD, un sistema multiagente “es una red de entidades capaces de solucionar problemas, que trabajan conjuntamente para encontrar respuesta a problemas que están más allá de la capacidad y el conocimiento individual de cada entidad” [53]. Recientemente, se le ha dado una connotación más general al término multiagente, y actualmente es utilizado para definir sistemas compuestos por múltiples componentes autónomos que poseen las siguientes características [53,62]:

a. Cada agente tiene capacidad para solucionar parcialmente el problema
b. No hay un sistema global de control

c. Los datos no están centralizados

d. La computación es asíncrona

e. Los agentes pueden ser heterogéneos, es decir, contar con arquitecturas o representaciones internas diferentes, comunicarse con diferentes lenguajes, etc.

Uno de los aspectos fundamentales en los SMAs tiene que ver con la interacción entre los agentes. Un agente es una entidad capaz de percibir su ambiente, procesar la información que percibe y generar un resultado esperado. La interacción está en la base de todos los procesos colectivos que se dan en los SMAs, e implica tareas de coordinación de acciones, negociación, comunicación, etc. Los tipos de coordinación de acción más conocidos son [30]:

- Sincronización de acciones: Este tipo de coordinación se centra en la definición de tareas para agentes distintos, los cuales deben llegar a puntos de encuentro en momentos específicos donde ya haya culminado la ejecución de las mismas. Aunque limitan la libertad de acción, la calidad de la coordinación es buena, evita conflictos, y permite un elevado número de agentes. Planificación: El fundamento de este tipo de coordinación es la subdivisión de tareas en subtareas y su distribución, pero ha de tenerse en cuenta la capacidad de replanificar. La libertad de acción está limitada, la calidad de la coordinación es muy buena, evita conflictos, permite un bajo número de agentes, y la cantidad de datos intercambiados es muy alta.

- Coordinación reactiva: La coordinación entre agentes reactivos se lleva a cabo mediante dos técnicas: la definición de un comportamiento reactivo en cada agente que favorezca la coordinación, y, la inclusión de marcas en el entorno que sean captadas por otros agentes.

Otro aspecto interesante en los SMAs es la característica de los agentes de inferir y mantener en el tiempo modelos cognitivos acerca de otros agentes. Los modelos cognitivos son aquellos que representan conceptos como intenciones, actitudes, deseos y habilidades [66]. Estos modelos cognitivos pueden funcionar como herramientas abstractas que proporcionen a cada agente formas para describir, explicar y predecir el comportamiento de los otros agentes. Al utilizarlos, cada agente podrá ser capaz de interactuar, negociar y adaptarse al grupo de la mejor manera posible, con lo que se podrían programar actividades colectivas [53,66].

2.3.2. Inteligencia Colectiva (IC):

La idea principal de la inteligencia colectiva (IC) sugiere que N agentes en una colonia cooperan para lograr alguna meta. Los agentes usan reglas simples para gobernar sus acciones, y por medio de las interacciones del grupo entero logran sus objetivos. Un tipo de auto-organización surge de la colección de acciones del grupo [46, 47].La IC resuelve problemas de manera flexible, adaptativa y descentralizada. La IC ha sido aplicada en distintas áreas como telecomunicaciones, robótica, transporte, aplicaciones militares, etc. [3,4,9,15,21,28,29,30,50,53,54,62,66,75,76,77].

La inspiración que sirvió de guía a los primeros desarrollos de sistemas basados en IC fueron las conductas colectivas de las colonias de insectos, y en particular, de las colonias de hormigas. Existen especies de hormigas que dejan rastros cuando realizan actividades de búsqueda, por ejemplo: cada una de las hormigas depositan una sustancia química, llamada feromona, cuando se trasladan desde una fuente de alimento a su nido. Así, el resto de las hormigas sigue esa sustancia para encontrar el camino más eficiente a una fuente de alimento. El proceso por el cual una hormiga es influenciada a ir hacia una determinada fuente de alimento, o por un sendero químico, por otra hormiga, es llamado reclutamiento [15]. En este caso, cada hormiga puede ser considerada un agente [15,50].

En la IC los agentes están agrupados en colonias, tal que cada uno de ellos pueda procesar información, modular su conducta de acuerdo a estímulos, y tomar la mejor decisión basada en la información del ambiente que les rodea. Pero el desafío más grande es hacer que los agentes trabajen de manera colectiva, que integren sus actividades individuales para generar resultados más complejos y eficaces. La IC se ha usado de diferentes maneras, y ha inspirado técnicas como optimización por enjambres de partículas, optimización usando sistemas artificiales de colonias de hormigas, entre otros [15,29,50,76].

En los estudios actuales de IC la conducta inteligente surge frecuentemente a través de la comunicación indirecta entre los agentes [15]. Veamos el caso de las hormigas. Individualmente, las hormigas son insectos de comportamiento simple con memoria limitada. Sin embargo, colectivamente las hormigas realizan tareas complicadas con un grado alto de consistencia. Algunos ejemplos de comportamiento sofisticado de las hormigas son: 1. Formación de puentes; 2. Construcción y mantenimiento de nidos; 3. Cooperación para cargar objetos grandes; 4. Búsqueda de la ruta más corta del nido a Fuentes de alimentos; 5. Regulación de la temperatura del nido; etc. [15,29,50,76].

En los modelos estudiados se han identificado dos tipos de comunicación indirecta, la primera involucra un cambio en las características físicas del ambiente. La construcción del nido es un ejemplo de esta forma de comunicación en que una hormiga observa el desarrollo de la estructura y agrega su pelota de barro a la cima de él [15]. La segunda está basada en *"señales"*. En este caso, algo es depositado en el ambiente que no hace ninguna contribución directa a la tarea, pero se usa para influir en la conducta subsiguiente [15]. La comunicación indirecta basada en señales está muy desarrollada en las hormigas, las cuales usan "feromonas" para proporcionar un sistema de señalización sofisticado. En general, una hormiga aislada se mueve esencialmente al azar, pero una hormiga que encuentra un sendero previamente transitado decidirá seguirlo con una probabilidad alta, y además reforzarlo con una cantidad de feromona. El comportamiento emergente colectivo de este proceso es una forma de comportamiento auto-catalítico, en el cual, mientras más hormigas siguen una ruta más feromona se deposita, de tal manera que otras hormigas sean más propensas a seguir la misma ruta. Este proceso es caracterizado por un ciclo de retroalimentación positiva, donde la probabilidad de que una hormiga escoja una ruta dada se incrementa a medida que otras hormigas hayan hecho la misma selección [15].

Uno de los principales trabajos en IC ha sido propuesto por Dorigo, Maniezzo y Colorni [15, 28, 29], donde modelan el comportamiento de una colonia de hormigas en búsqueda de alimentos". El agente (hormiga) individualmente no posee capacidad para resolver el problema, el conocimiento y la inteligencia se generan como resultado de acciones sociales del grupo de agentes. Ese modelo ha sido aplicado, entre otros, al problema del viajero de comercio [3, 28]. En este caso, el objetivo es encontrar la ruta más corta que conecte dos ciudades. Cada ciudad debe ser visitada solo una vez. Sea d_{ij} la distancia entre las ciudades *i* y *j*, *x* e *y* las coordenadas de cada ciudad; d_{ij} puede ser definido como sigue:

$$d_{ij} = \left[(x_i - x_j)^2 + (y_i - y_j)^2\right]^{1/2} \qquad (1)$$

La regla de transición, que es la probabilidad de que una hormiga *k* vaya de la ciudad *i* a la ciudad *j*mientras se construye la ruta t^{th}, está dada por:

$$P_{ij}^{k}(t) = \left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta} \Big/ \sum_{l \in J_i^k}\left[\tau_{il}(t)\right]^{\alpha}\left[\eta_{il}\right]^{\beta} \qquad (2)$$

Donde α y β son dos parámetros ajustables que controlan el peso relativo entre la intensidad del rastro de feromona $\tau_{ij}(t)$, y la visibilidad η_{ij} (este es el inverso de d_{ij}). Si α=0 entonces la ciudad más cercana será más probable que sea seleccionada, esto corresponde a un algoritmo estocástico clásico. Si por el contrario β=0, el feromona es el que guía el proceso de selección.

Luego de completar una ruta, cada hormiga k deja una cantidad de feromona $\Delta\tau_{ij}^{k}(t)$ sobre cada camino (i,j) que ha visitado; el valor $\Delta\tau_{ij}^{k}(t)$ depende del rendimiento de cada hormiga. La cantidad de feromona de cada camino se actualiza según la siguiente expresión:

$$\tau_{ij}(t) \leftarrow (1-\rho)*\tau_{ij}(t) + \Delta\tau_{ij}(t) \qquad (3)$$

donde $\Delta\tau_{ij}(t) = \sum_{k=1}^{m}\Delta\tau_{kij}(t)$, m es el número de hormigas, y ρ es la rata de evaporación del feromona. El monto inicial de feromona sobre un camino es una pequeña constante positiva.

Se asume que el número total de hormigas *m* es una constante en el tiempo. Este es un parámetro importante, pues muchas hormigas podrían reforzar rápidamente caminos no óptimos, y generar convergencia de soluciones incorrectas. Por otro lado, muy pocas hormigas podrían no producir los efectos esperados de cooperación, pues los montos de feromona serían muy pequeños.

En los trabajos de Dorigo, Maniezzo y Colorni [15, 28, 29], ellos han propuesto tres tipos de algoritmos de hormigas, denominados [15]: Densidad de hormigas, Cantidad de hormigas, y ciclo de hormigas, los cuales pueden ser usados para resolver diferentes problemas de optimización combinatoria. Estos algoritmos proponen una manera específica de cálculo de $\Delta\tau_{ij}^{k}(t)$, los cuales son [21]:

- En el cálculo de la cantidad de hormigas, una constante de feromona Q_1 es depositada en la ruta ij cada vez que la hormiga se desplaza de i a j. La fórmula es:

  $\Delta\tau_{ij}^{k}(t) = \frac{Q_1}{di_j}$ Si la hormiga k se desplaza de i a j entre t y t+1,

  de lo contrario $\Delta\tau_{ij}^{k}(t) = 0$

- En el modelo de densidad de hormigas, cada hormiga que se desplaza de i a j deja Q_2 unidades de feromona por cada unidad de medida del trayecto. La fórmula matemática se expresa como sigue:

  $\Delta\tau_{ij}^{k}(t) = Q_2$ Si la hormiga k se desplaza de i a j entre t y t+1

  de lo contrario $\Delta\tau_{ij}^{k}(t) = 0$

- En el ciclo de hormigas se introduce una diferencia mayor con respecto a las anteriores, la misma consiste en que $\Delta\tau_{ij}^{k}(t)$ se calcula luego de que todo el recorrido se ha completado, la fórmula matemática es la siguiente:

$\Delta\tau_{ij}^{k}(t) = \frac{Q_3}{L^k}$ Si hormiga k usa la ruta de i a j en su recorrido, de lo contrario $\Delta\tau_{ij}^{k}(t) = 0$. L^k es la longitud del recorrido hecho por la hormiga k.

Estos algoritmos, y extensiones a los mismos [3, 4, 15, 28, 50, 29] han sido usados para resolver problemas de asignación cuadrática, de enrutamiento, entre otros.

2.4. Computación Reflexiva (CR):

La reflexión orientada a lenguajes de programación es un paradigma que se origina del trabajo del Brian Smith, orientado a la auto-conciencia y auto-referencia de los sistemas o programas, lo cual les permite cambiar su "comportamiento" según los requerimientos y necesidades del entorno de ejecución [11,31]. La auto-representación de los sistemas reflexivos les permite observarse, "razonar" sobre sí mismos, y

modificarse.

Resumiendo lo anterior, la reflexión es la habilidad de un programa en ejecución de examinarse a si mismo, y eventualmente, también a su ambiente, con la finalidad de ejecutar cambios en su comportamiento según sea necesario para auto-mantenerse. Para realizar esto, se requieren de varios conceptos que presentamos a continuación.

Introspección: Es la habilidad de un programa de observar y razonar sobre su estado. El término introspección es utilizado en el campo de la filosofía para describir los métodos utilizados por la mente para aprender acerca de si mismo, y producir modelos y teorías del subconsciente [11,31].

Intercesión: Es la capacidad del programa de modificar su propio estado de ejecución, representación o comportamiento. Por medio de la intercesión es posible modificar la secuencia de ejecución de un sistema, sin que sea necesario realizar cambios en el código de la aplicación [11,31].

En los sistemas reflexivos existe una parte orientada al programa o aplicación como tal, y una parte reflexiva. En la aplicación se genera información que usa la parte reflexiva para ejecutar las tareas de Introspección e Intercesión. La actividad fundamental de la parte del programa o aplicación es resolver problemas y retornar información sobre su ejecución. La parte reflexiva de la aplicación se centra en manejar información sobre el programa en sí y su funcionamiento, pudiendo acceder y manipular dinámicamente su representación y comportamiento, aún en tiempo de ejecución, sin ser necesaria la redefinición del código fuente inicial [11,31].

Un sistema reflexivo orientado a objetos es estructurado en dos o más niveles (fig.5), formando una "*Torre Reflexiva*". El primer nivel, o *nivel base*, describe los cómputos o procesos que el sistema debe realizar, y contiene los objetos que resuelven un problema dado. Usualmente, el nivel base, o nivel 0, corresponde a la aplicación como tal [11,31]. El *nivel meta* describe como se deben realizar los cómputos o procedimientos incorporados en el nivel base, y verifica que su funcionamiento sea el esperado o requerido. Este nivel está formado por objetos que desarrollan computación sobre la aplicación, para realizar las tareas de Introspección e Intercesión [11,31] (parte reflexiva). Los objetos o entidades que se encuentran en el nivel base son denominados *entidades base,* y las entidades u objetos que se encuentran en el nivel meta son denominados *entidades meta* [11,31].

La jerarquía de dos niveles, base y meta, puede ser generalizada en n niveles, y cualquier cambio en los

niveles inferiores es reflejado en los niveles superiores correspondientes. En una torre de n niveles de reflexión, el nivel 0 es el programa o aplicación principal (nivel base), el nivel 1 es el intérprete del nivel 0, y así sucesivamente, hasta un nivel n+1, el cual sería el intérprete del nivel n (fig. 5) [11].

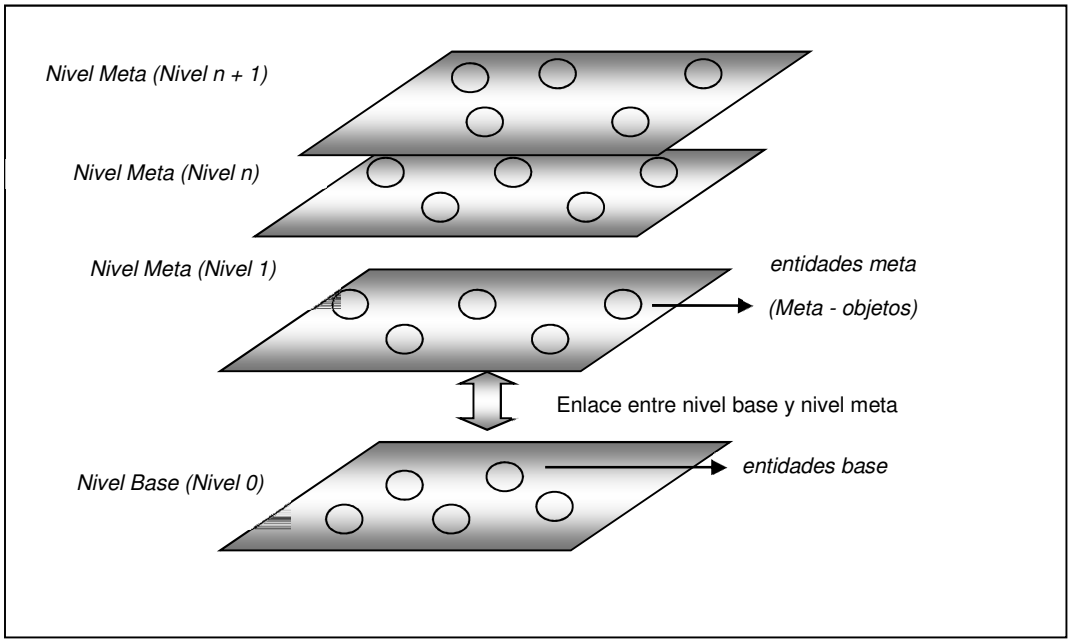

Figura 5: Arquitectura Reflexiva

Los niveles meta están compuestos por meta–objetos, y son utilizados generalmente para describir y explicar las estructuras y comportamiento de los programas en termino de sus propias estructuras de datos y control [11,31]. Esto permite que un nivel meta pueda manipular las aplicaciones de un nivel inferior; así, en una torre reflexiva cada nivel puede ser manipulado por la capa o nivel inmediatamente superior. Los niveles están conectados de forma causal, de manera tal que los cambios en el nivel base son reflejados en el nivel meta. Idealmente, el nivel meta tiene acceso al nivel base, pero el base no tiene conocimiento de la existencia del meta. Esto permite aislar el comportamiento de los objetos del nivel base de los mecanismos adaptativos existentes en el nivel meta.

Un Protocolo de Meta-Objetos (MOP) provee un mecanismo para representar la información del sistema como dato, para que la información pueda ser manipulada en el nivel meta. Así, el comportamiento computacional del nivel base es transformado en dato, esta actividad es denominada

"reificación". Las entidades reificadas constituyen la meta-información, a través de las cuales es posible desarrollar comportamientos reflexivos. Por ejemplo, en la fig.6 "perro" es una clase de un objeto, y representa un metaobjeto de clase "perro". El objeto "fido" es una instancia de un perro que opera en la aplicación [31].

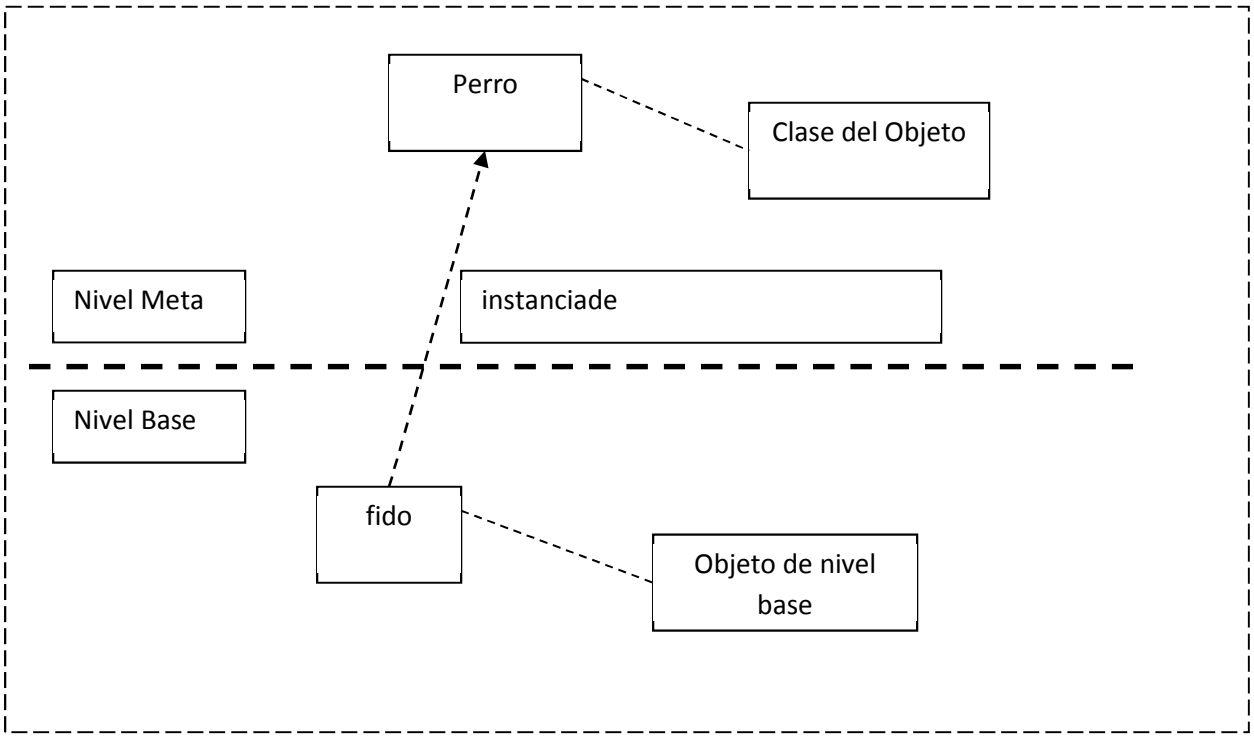

Figura 6: La relación instancia de es representada por una dependencia, la cual conecta objetos del nivel base a sus clases en el nivel meta

El mecanismo utilizado más comúnmente para activar meta-objetos es por medio de la intercepción de mensajes entre objetos. En la intercepción de mensajes, cualquier mensaje enviado entre objetos es delegado al meta-objeto asociado. Un mensaje puede ser interceptado cuando el mensaje es enviado o recibido. Independientemente de esta decisión de implementación, el protocolo de meta-objetos debe proveer mecanismos para recuperar información sobre el objeto que envió el mensaje, el método correspondiente al mensaje, y sus argumentos.

El implementar las propiedades de la computación reflexiva sobre el paradigma de orientación a objetos

presenta los siguientes beneficios, comparado con otros paradigmas de programación [46]:

- Granularidad. Se refiere a que cualquier objeto puede ser reflejado, es decir, se puede construir un meta objeto a partir de cualquier objeto base, o incluso a partir de otro meta objeto, de esta manera es posible, por ejemplo, reflejar objetos que representan abstracciones de otros objetos.
- Modularidad. La reflexión toma lugar de una manera local (en el objeto en específico que se desee). Esto es, las modificaciones necesarias para incorporar comportamiento reflexivo se necesitan hacer solamente localmente. Además, es posible incorporar comportamiento reflexivo sin necesidad de modificar el código fuente. Por otro lado, solamente la interpretación del objeto reflejado es afectada por el mecanismo de reflexión, los otros objetos de la misma clase son tratados en una forma normal. Esto permite a los programadores definir diferente comportamiento reflexivo para cada objeto.
- Introspección. La mayoría de los lenguajes orientados a objetos introducen introspección, por ejemplo, manipulación por defecto, razonamiento sobre cambios, y múltiples vistas de los objetos. Es importante representar esta información en una manera uniforme y explícita, lo cual produce mayor poder expresivo y una simplificación de la semántica del lenguaje.
- Seguimiento (tracing). Un lenguaje orientado a objetos reflexivo es adaptable para realizar un seguimiento de las actividades realizadas por el sistema base. Es posible incorporar temporalmente comportamiento reflexivo a un objeto, observar lo que pasa durante su tiempo de vida, etc.

Es posible incorporar el concepto de meta-objetos y las características reflexivas en los lenguajes de programación orientados a objetos implementando los siguientes mecanismos [46]:

- representación de las propiedades y comportamiento de objetos y clases como dato;
- intercepción de mensajes entre objetos y activación del correspondiente meta-objeto que administra dicho mensaje.

La complejidad en la implementación de estos mecanismos depende de las características del lenguaje de programación. Varios lenguajes orientados a objetos han sido extendidos para soportar el concepto de meta-objetos. Por ejemplo, extensiones de LISP son 3-KRS, CLOS; extensiones de C++ son Open C++

versión 1 y versión 2, y Mirror entre otros [46, 69, 78]. Para implementar estos mecanismos en lenguajes compilados, como C++ y Java, el cual es un lenguaje totalmente orientado a objetos, se realiza usualmente un pre-procesamiento del código fuente. Este pre-procesamiento agrega mecanismos de intercepción de mensajes, los cuales informan a los meta-objetos de los mensajes enviados en el nivel base.

Resumiendo todo lo anterior, la computación reflexiva trabaja en base a programas que podríamos llamar “supervisores” o “modificadores”, los cuales están enlazados a la aplicación de tal manera que sus procesos, entradas y salidas puedan ser evaluadas permanentemente por ellos, para generar las modificaciones necesarias a la aplicación solo si fuese necesario. Estos programas pueden ser definidos usando objetos, y cada objeto supervisor puede tener a su vez otro objeto supervisor ubicado en una capa superior, formando una “torre reflexiva” (ver fig. 5). Así, los objetos programados bajo computación reflexiva tienen la particularidad de que pueden ser no solo supervisados, sino manipulados, cambiando sus comportamientos en tiempo de ejecución sin que una nueva compilación o redefinición del código fuente sea necesaria.

Un punto a considerar al implementar reflexión computacional es el rendimiento. En específico, los recursos que se requieren para analizar la información relacionada a la aplicación base, redireccionar dicha información al nivel meta, ejecutar acciones sobre el nivel base, entre otras cosas, pueden introducir costos altos en el sistema que repercuten en el rendimiento del mismo [46].

Capítulo 3

Búsqueda y Selección de Componentes de Software

3.1 Bases Teóricas:

Para el diseño e implementación de los algoritmos inteligentes de búsqueda y selección de componentes se han tomado en consideración dos áreas de la computación, como son la Inteligencia Artificial Colectiva y la Ingeniería de Software Orientada a Componentes.

En este capítulo proponemos dos algoritmos, basados en agentes inteligentes, para los dos primeros problemas, es decir los problemas de búsqueda y selección de componentes. Los agentes inteligentes pueden ser utilizados para resolver problemas sofisticados, como la búsqueda y selección de componentes de software que pueden estar ubicados en diferentes repositorios e integrados para generar sistemas de software completos. Estos problemas son altamente complejos, entre otras cosas, debido a que los componentes de software no tienen siempre una correspondencia perfecta con los componentes que se están buscando, y si la tuvieran, entonces se debe asegurar la posible integración de los componentes, así como la posibilidad de mantener la seguridad de la aplicación y el rendimiento.

Respecto al problema de búsqueda, existen propuestas para la búsqueda de objetos dentro de ciertos modelos (como RM-ODP) o plataformas (CORBA) [14]. También comienzan a aparecer herramientas específicas para componentes, como por ejemplo el denominado “COTStrader" (www.cotstrader.com). Dicha herramienta permite buscar y seleccionar componentes a partir de la definición en plantillas XML de su funcionalidad y otras propiedades de calidad. El segundo problema se centra en los procesos y herramientas para la evaluación y selección de componentes. Además de tener en cuenta los requisitos funcionales de la aplicación, es necesario considerar otros factores que también intervienen a la hora de seleccionar componentes. El problema es que este tipo de requisitos, denominados “extra-funcionales”, son difíciles de evaluar. Este tipo de factores priman muchas veces, incluso más que los funcionales, pues un diseñador es capaz de adaptar la arquitectura de un sistema para incluir en ella un componente deseado, o bien para evitar la presencia de un componente de un fabricante en el cual no se confía.

Como se dijo antes, este trabajo propone dos algoritmos estocásticos para la búsqueda y selección de componentes de software, los cuales están inspirados en la teoría de Inteligencia Colectiva. Estos

algoritmos permiten buscar un grupo de componentes con especificaciones particulares en internet, y seleccionar el mejor componente de entre ellos según dichas especificaciones. Estos algoritmos utilizan trazas de feromona para identificar los mejores componentes. El valor de feromona de cada componente se incrementa o disminuye cada vez que el componente es probado.

En este trabajo se utiliza un archivo para caracterizar a cada componente, al cual se le denomina “Perfil del Componente”. Este es un archivo XML en el que no solo se almacena información asociada a la calidad del componente, sino que también, de manera general contiene tanto información estática como dinámica que describe aspectos funcionales y no funcionales del componente (ver figura 7). Entre la información estática se incluyen datos como identificador del componente, nombre del componente, que representan aspectos no funcionales, y cualquier otra información que sea permanente en el tiempo. Entre el grupo de datos dinámicos se tiene información relacionada a aspectos funcionales como las dependencias del componente, sistema operativo y lenguaje de programación donde ha sido usado, y aspectos no funcionales como última fecha de actualización, etc. Además, se incluye una variable que permitirá al algoritmo de selección hacer su trabajo de manera más eficiente, esta es la variable “feromona”. Toda esta información permite determinar si el componente podrá ser integrado a un sistema dado, según los requerimientos establecidos en él.

Un componente puede tener tantos perfiles como sea requerido, cada uno de los cuales está asociado a un dominio en particular donde ha sido usado el componente, de tal manera que en el perfil se describe su comportamiento en ese dominio como por ejemplo el sistema operativo en el que funciona el componente, el lenguaje de programación que utiliza, tiempo de ejecución promedio, seguridad asociada, rendimiento permitido, etc. Como se dijo antes, también habrá una feromona por cada perfil, asociada al rendimiento del componente en ese dominio en particular. Cada vez que un componente ha sido seleccionado o rechazado, los algoritmos de selección que se presentan en este trabajo actualizan la feromona de uno de sus perfiles (el vinculado al dominio de aplicación en particular). Si es rechazado, una taza de evaporación se aplica sobre el feromona; si es seleccionado, se actualiza el feromona de acuerdo al rendimiento del componente, pudiendo este valor incrementarse o decrementarse. Así, esta variable se incrementa o decrementa según la eficiencia del componente. A medida que este valor sea mayor, el componente tiene más probabilidades de ser seleccionado entre un grupo de componentes que cumplen con los mismos requerimientos.

Los dos algoritmos de búsqueda y selección de componentes de software propuestos los llamaremos Algoritmo A (AA) y Algoritmo B (AB). AA utiliza un agente para cada componente que se desea seleccionar, esto significa que AA crea tantos agentes como componentes de software se deseen buscar y seleccionar; por otra parte, AB utiliza tantos agentes como se desee, independientemente del número de componentes que se vayan a buscar y seleccionar, esto es debido a que cada agente del AB está encargado de buscar y seleccionar todos los componentes requeridos.

```
<componentInformation>
<uniqueID>PRU0707</uniqueID>
<name>Report Manager</name>
<license>gpl</license>
<componentInformation>
<sub-profile1>
<pheromone>0.1</pheromone>
<location> www.cemisid.ula.ve/components/rmc </location>
<os>debian</os>
<dependency>string.dll - math.dll - proactive.jar - fractal.jar
- examples.jar
</dependency >
<ms>3</ms>
<et>0.01</et>
</sub-profile1>
```

Figura 7: Perfil de un Componente (Archivo XML)

Los requerimientos generales para ambos algoritmos son:

- Si el componente a seleccionar no existe o no se encuentra, AA y AB generan un mensaje de alerta.
- Cada componente posee por lo menos un perfil en un archivo XML, y ese archivo debe permitir la incorporación de otros perfiles.
- Se asume que los requerimientos iniciales contienen la información necesaria sobre los componentes para su búsqueda.

3.2 Modelo AA

Este algoritmo está conformado por un agente creado por cada componente a buscar y seleccionar, por lo tanto, deben crearse tantos agentes como componentes se requieren. El agente tendrá la tarea de seleccionar solo un componente de todo el grupo de componentes que haya encontrado. Las etapas que componen el proceso de selección son las siguientes:

1. El agente realiza un recorrido para el componente solicitado.
   1.1. La trayectoria seguida en cada recorrido consiste en visitar, aleatoriamente, 1, 2 ó n repositorios que contienen componentes de software.
   1.2. Al final de la trayectoria seguida, en cada uno de los recorridos realizados, el agente habrá acumulado un grupo de componentes, en un conjunto llamado cc, asociados al componente relacionado al recorrido, candidatos a ser seleccionados.
2. Finalmente, el agente realiza la selección del mejor componente encontrado, del grupo de componentes conseguido que denominamos cc.

El ensamblaje de los componentes estará a cargo del programador, luego de que cada agente complete su proceso de selección y se tengan a disposición todos los componentes requeridos (ver fig. 8). El programador incorpora a la aplicación a desarrollar los componentes seleccionados. Después, el agente actualiza la traza de feromona de los componentes.

La ecuación utilizada para establecer la relación entre el componente ideal y el componente evaluado es:

$$X_{lju}^{k} = 1 + f\,(C_k, N_{lju}^{k}) \qquad (4)$$

donde X_{lju}^{k} es el grado de correlación entre el perfil deseado del componente k y el componente j, el cual está en el repositorio l, y ha sido pre-seleccionado por el agente k porque es similar al componente ideal. C_k identifica las características ideales del componente k deseado (por ejemplo, el sistema operativo, el lenguaje de programación, máximo número de dependencias, entre otras cosas). N representa las características reales encontradas en el componente j (específicamente, su perfil u corresponde a la plataforma donde será usado el componente j en la aplicación que se está

desarrollando); y la función f calcula las diferencias entre las características deseadas y las reales entre los componentes k y j. Si X_{lju}^{k} está cercano a 1, significa que las características ideales son similares a las requeridas, mientras menor sea el valor de X_{lju}^{k} mejor es la correlación entre los componentes comparados.

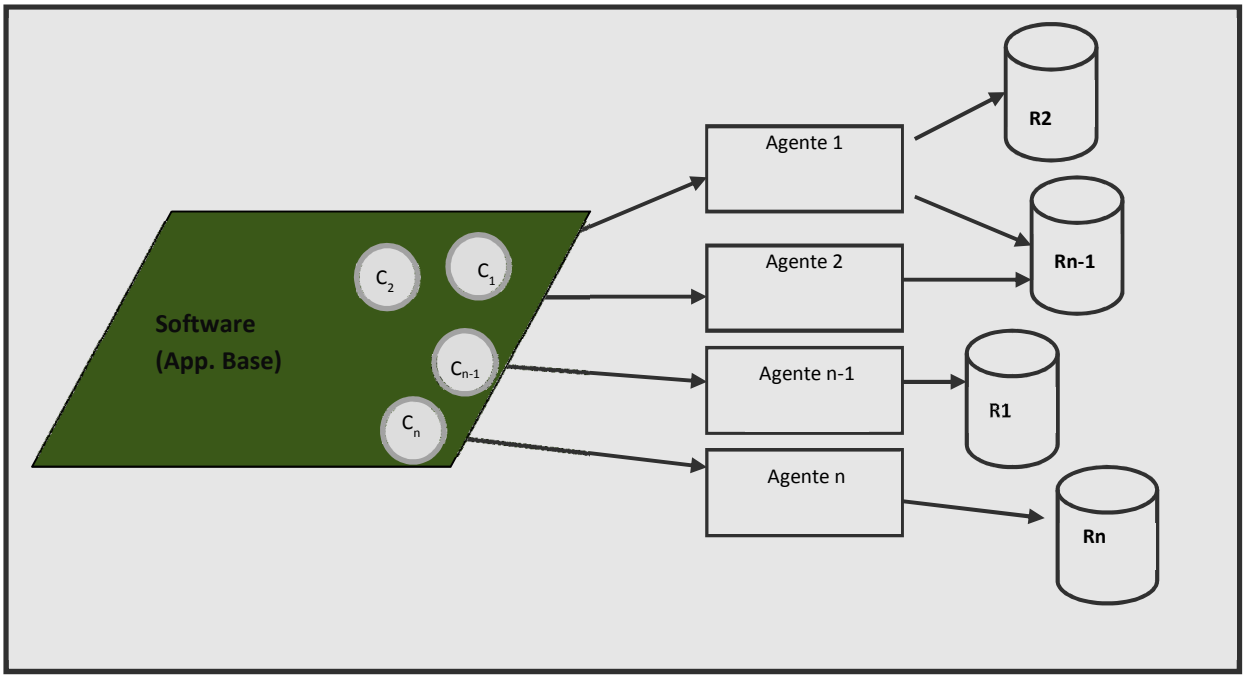

Figura 8: Algoritmo A – Un Agente por cada componente

Cada agente selecciona un componente bajo las siguientes premisas [2,5,6,7,15,21,28,29,70]:

- El valor ideal de X_{lju}^{k} es 1, y se obtiene cuando una correlación entre dos componentes es perfecta.
- El monto de feromona $Y_{lju}(t)$ es relativa a cada componente i, que es ubicado en un repositorio l para un perfil u, el cual será actualizado luego de que el componente es utilizado.
- El monto de feromona se incrementa dependiendo del rendimiento del componente.
- El valor de feromona de todos los componentes encontrados que no fueron seleccionados, se decrementa según una rata de evaporación (ver ecuación 3).

La ecuación de transición que calcula la probabilidad para que un agente k seleccione un componente j, que posee un perfil u, en un repositorio l, entre un grupo cc de componentes, en una iteración t, se

calcula como sigue [2,5,6,7,15,21,28,29,70]:

$$P_{lju}^{k}(t) = \left[\tau_{lju}(t)\right]\left[X^{k}{}_{lju}\right]^{-1} \Big/ \sum_{rsn}\left[\tau_{rsn}(t)\right]\left[X^{k}{}_{rsn}\right]^{-1}$$

$\forall$rsn$\in$ cc (normalmente cuando X^{k}_{rsn} es cercano a 1) (5)

Esto corresponde a un algoritmo estocástico. Luego de completar la selección y pruebas de los componentes, cada agente actualiza una cantidad de feromona $\Delta\tau_{iju}^{k}(t)$, cuyo valor depende del rendimiento de la aplicación en el sistema (R) donde el componente es utilizado.

$$\Delta\tau_{lju}^{k}(t) = (\mathrm{X}_{\mathrm{lju}}^{\mathrm{k}} * R)^{-1} \qquad (6)$$

donde

$$R = f(ET, M, ND) \qquad (7)$$

La función de rendimiento viene dada por el tiempo de ejecución (ET) de la aplicación, la cantidad de memoria utilizada (M), y el número de dependencias[6] (ND) que el componente necesita para su implementación en la plataforma específica donde será usado.

En nuestro modelo es necesario el cálculo de la retroalimentación positiva y negativa, esta última, también llamada, evaporación de feromona, es incorporada a través de un coeficiente de evaporación ρ (ver ecuación 3) [2,5,6,7,15,21,28,29,70].

Finalmente, para determinar el proceso de selección de un componente la siguiente regla es definida:

[6]Se entiende por dependencias a componentes que requieren de otros componentes para funcionar. Es decir, componentes de software que dependen de otros para ejecutarse. Por tanto, cualquier cambio que se realice a algún componente también afecta a todas sus dependencias.

$$S^*_k = \begin{cases} \max - \arg_{rsn \in CC} \left\{ P^k_{rsn} \right\} \\ J \quad \text{(aleatorio)} \end{cases} \qquad (8)$$

donde el valor de P es dado por la ecuación 5, S^*_kes el componente seleccionado, y J es un valor aleatorio para seleccionar un componente. Lo que expresa la ecuación 8 es que algunas veces es usada la ecuación 5 para seleccionar un componente, y otras veces se escoge aleatoriamente. Esto último permite que eventualmente nuevos componentes sean seleccionados.

El algoritmo base, y los dos macro-algoritmos de este modelo, son [2]:

Algoritmo Base:

1. Definición de la arquitectura del software.
   1.1. Definición e identificación de los perfiles del componente(s) requerido(s) k.
2. Llamar al algoritmo AA.
3. Ensamblar el sistema (lo hace el diseñador del sistema).
4. Analizar el rendimiento del sistema (R) utilizando la Ecuación 7.
5. Actualizar la traza de feromona para todos los componentes utilizando las ecuaciones 3 y 6.

Algoritmo de Búsqueda y Selección AA:

1. Crear k agentes inteligentes de selección, uno por cada componente a buscar.
2. Inicializar cada agente.
3. Repita para cada agente i=1 a k, donde k es el número total de componentes a buscar y seleccionar.
   3.1. Identificar el grupo de posibles componentes a utilizar (cc) utilizando la ecuación 4 (Buscar los componentes deseados utilizando los repositorios disponibles, por ejemplo: Freshmeat, Sourceforge, etc.)
   3.2. Seleccionar componente i usando Ec. 8

3.3 Modelo de Selección AB

En este caso, el sistema utiliza tantos agentes como desee, y cada uno debe seleccionar todo el grupo de componentes que el sistema a desarrollar requiere.

Las etapas que componen el proceso de selección son las siguientes:

1. Cada agente realiza un recorrido por componente solicitado, independiente al seguido por el resto de los agentes, en busca del componente.
   1.1. La trayectoria seguida en cada recorrido consiste en visitar, aleatoriamente, 1, 2 ó n repositorios que contienen componentes.
   1.2. Al final de la trayectoria seguida, en cada uno de los recorridos realizados, el agente k habrá acumulado un grupo de componentes, llamado cc_{ik}, asociados al componente i relacionado al recorrido, candidatos a ser seleccionados.
2. Finalmente, cada agente realiza la selección del mejor componente encontrado, del grupo de componentes cc conseguido, por componente solicitado.

El ensamblaje de los componentes está a cargo del programador, luego de que cada agente creado complete todo el proceso de selección y tenga a disposición todos los componentes requeridos (Ver Fig. 9). El programador selecciona, según algún criterio, los componentes que requiere del grupo de componentes seleccionado por los diferentes agentes, y los incorpora a la aplicación a desarrollar.

Después, los agentes actualizan la traza de feromona de los componentes.

De la ecuación (4), y sustituyendo $f(C_k, N_{lju}^k)$ por $\Sigma H_i - \Sigma N_{lju}$, la fórmula para establecer el grado de correspondencia entre el componente ideal y el componente preseleccionado se define como sigue:

$$X_{lju}^i = 1 + \Sigma H_i - \Sigma N_{lju} \qquad (9)$$

donde X_{lju}^i es el grado de correspondencia entre el componente ideal i, y el componente j ubicado en el repositorio l. Σ Hi identifica la sumatoria de las características ideales que debe tener el perfil del

componente i deseado (por ejemplo, sistema operativo, lenguaje de programación, número máximo de dependencias, entre otras). Por otro lado, Σ N representa la sumatoria de las características reales encontradas, similares a las deseadas, del componente preseleccionado (candidato), cuyo perfil u se corresponde a la plataforma donde será usada la aplicación en construcción. Mientras más cercano a 1 sea el valor de X_{lju}^{i}, mayor será la similitud entre las características ideales y las características reales del componente j encontrado.

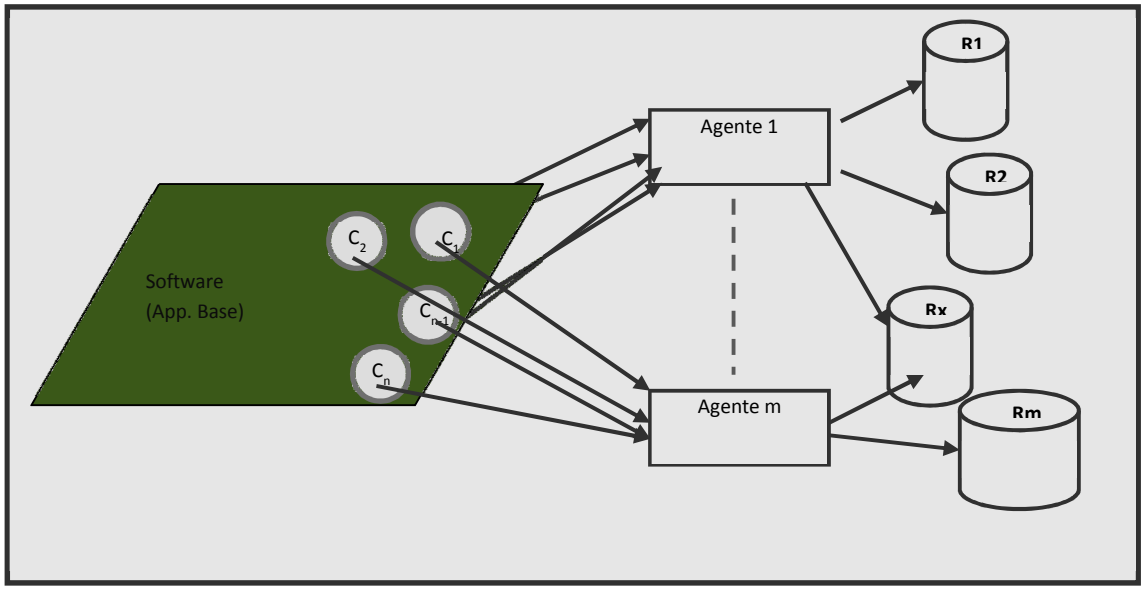

Figura 9: Modelo de Selección. Cada agente busca el grupo de componentes a utilizar

Cada agente evalúa un conjunto de componente de software, por cada componente requerido, considerando [2,5,6,7,15,21,28,29,70]:

1. El valor ideal de X_{lju}^{i} es 1, el agente escoge componentes cuya correspondencia entre el componente deseado y el encontrado es cercana a ese valor.
2. El monto de feromona Y_{lju} (t) relacionado a cada componente j con perfil u, ubicado en el repositorio l, que conforman cada uno de esos conjuntos, será actualizado luego de que el usuario escoja los componentes a ensamblar en la aplicación y los pruebe en ella. El monto se incrementará o decrecerá, dependiendo del rendimiento de la aplicación, y de la tasa de

evaporación.

La ecuación de transición que calcula la probabilidad de que un agente k seleccione un componente j con perfil u, ubicado en el repositorio l, de entre un grupo cc_{ik} de posibles componentes a seleccionar, es [2]:

$$P_{lju}^{ik}(t) = \frac{\left[Y_{lju}(t)\right]\left[X^{i}{}_{lju}\right]^{-1}}{\sum_{rsn}\left[Y_{rsn}(t)\right]\left[X^{i}{}_{rsn}\right]^{-1}} \qquad (10)$$

$\forall rsn \in cc_{ik}$(normalmente cuando X^{k}_{rsn} es cercano a 1)

donde $Y_{lju}(t)$ representa la cantidad de feromona para el componente j que ha sido encontrado por el agente k en el repositorio l, cuyo perfil u se corresponde a la plataforma donde será usada la aplicación. Los agentes son creados e inician el proceso de selección al azar. Es importante notar que el valor de la probabilidad $P_{lju}^{ik}(t)$ pudiera ser diferente para dos agentes evaluando un mismo componente, ya que este depende del grupo de componentes cc_{ik} que cada agente ha encontrado.

Como se dijo anteriormente, cada agente k deposita una cantidad de feromona $\Delta Y_{lju}(t)$ en cada uno de los componentes seleccionado por el usuario, esa cantidad depende del rendimiento R de la aplicación en el sistema donde el componente es usado [2, 15]:

$$\Delta Y_{lju}(t) = (X^{k}_{lju} * R_{lju})^{-1} \qquad (11)$$

Tal que

$$R_{lju} = f(TE, M, ND_{lju}) \qquad (12)$$

La función de rendimiento (R_{lju}) viene dada por el Tiempo de Ejecución (TE) de la aplicación en el

sistema donde ella es usada, la cantidad de memoria utilizada (M), y el número de dependencias $(ND)_{lju}$ que el componente tiene.

En nuestra propuesta es necesaria la retroalimentación positiva y negativa. Esta última se hace a través de la tasa de evaporación de feromona, ya sea que el componente sea seleccionado por el programador o no. En general, la retroalimentación positiva y negativa es realizada a través de la actualización de la traza como sigue [15]:

$$Y_{lju}(t) = (1 - \rho) * Y_{lju}(t) + \Delta Y_{lju}^{k}(t) \qquad (13)$$

El monto inicial de feromona de los componentes se asume como un número aleatorio. ρes una constante que controla la evaporación de la traza. Finalmente, para determinar el componente "i" a seleccionar se define la siguiente regla de transición:

$$\mathbf{S^*}_{ki} = \begin{cases} \arg\max_{rsn \in cc_{ik}} \left\{ P_{rsn}^{ik} \right\} \\ \text{J} \quad (\text{Valor Aleatorio}) \end{cases} \qquad (14)$$

donde el valor de P_{rsn}^{ik} viene dado por la Ec.(10). Así, $\mathbf{S^*}_{ki}$ es el componente i seleccionado por el agente k de un grupo de componentes cc_{ik}, ó, es un componente J seleccionado aleatoriamente.

Algoritmo de Búsqueda y Selección AB es:

1. Definición e identificación de los componentes requeridos y sus respectivos perfiles deseados.
2. Crear k agentes de selección
3. Cada agente realiza la selección de los componentes requeridos usando el procedimiento "selección".
4. Seleccionar un componente de entre la selección hecha por los distintos agentes, ensamblar al sistema y ejecutarlo.

5. Actualizar la traza de feromona para cada perfil de componente de los distintos cc_{ik}.

Procedimiento “selección”:

1. Repita desde i = 1 hasta n (n es el número de componentes deseados)
   1.1. Búsqueda de componentes similares al componente i requerido (estos componentes conforman al conjunto cc_{ik}).
   1.2. Seleccionar uno de ellos usando (8).

En el capítulo 4 se explica el basamento teórico y práctico del Middlware Reflexivo Colectivo el cual integra los algoritmos de búsqueda y selección de componentes de software que se han definido en este capítulo.

Capítulo 4

Middleware Reflexivo Colectivo

4.1 Bases Teóricas

Además de los aspectos relacionados con computación reflexiva, ingeniería de software y arquitectura de software que ya han sido mencionados en los capítulos 1 y 2, la arquitectura de middleware reflexivo que esta tesis ha desarrollado tiene una base importante proveniente de la teoría de los sistemas autónomos.

Un software autónomo es aquel capaz de auto-manejarse, el cual requiere conocimiento de sus componentes tal como el estado de cada uno de ellos, sus capacidades, etc. Los primeros trabajos en esta área se generan en el año 2001 por IBM y en su mayoría están orientados a problemas de enrutamiento [45]. Un sistema autónomo es consciente de las condiciones de su ambiente y el contexto que lo rodea. En este sentido, este tipo de sistemas puede generar cambios proactivos o predecir comportamientos a futuro. Todo esto provee oportunidades para planear y afectar el estado del sistema, si esto fuese necesario [57]. Las características de los sistemas autónomos han sido aplicadas en cuatro áreas fundamentales:

1. Capacidades de auto-configuración: Adaptarse a condiciones impredecibles, a través de cambios automáticos en su configuración.
2. Capacidades de auto-reparación: Prevención y recuperación de fallas.
3. Capacidades de auto-optimización: Auto-regulación permanente para mejorar su funcionamiento (ser cada vez más eficiente, etc.).
4. Capacidad de auto-protección: Identificación y defensa contra diferentes tipos de ataques, tales como virus, accesos no autorizados, y negación de servicios, cuando están bajo ataque.

Una arquitectura basada en computación autónoma incluye cuatro aspectos [45]:

- La definición de procesos, la cual describe los procesos de negocios[7] que están automatizados en el sistema autónomo.
- La definición de los tipos de recursos que serán utilizados y manejados por el sistema autónomo.
- Las características técnicas de la arquitectura, la cual describe como los elementos del sistema serán integrados para soportar los servicios ofrecidos por la aplicación.
- Los patrones de comportamiento de referencia deseados del sistema, que describen los resultados esperados del sistema para situaciones reales concretas, los cuales normalmente son los casos comunes que se encuentran en la realidad donde funcionará el sistema.

4.2 Descripción conceptual del Middleware reflexivo:

Es una arquitectura compuesta de tres agentes inteligentes, para monitorear y modificar sistemas desarrollados con el uso de componentes de software. Una configuración inicial es requerida para el buen funcionamiento del middleware, la misma es realizada utilizando un archivo XML pre-definido, según se muestra en el anexo 1. Las decisiones que se toman a nivel del middleware son centralizadas en el agente Reflector descrito más adelante. Por otro lado, en la implementación que se hizo el grupo de agentes que conforman dicho middleware residen también centralizadamente en un mismo procesador. Ahora bien, en una aplicación distribuida existiría una replicación de los 3 agentes en cada sitio, y ellos realizarían las tareas respectivas definidas para ellos en el middleware reflexivo para los componentes de la aplicación distribuida ubicados en cada sitio. Los agentes inteligentes principales del Middleware Reflexivo se describen a continuación (ver fig. 10):

Monitor: Su meta principal es observar (introspección) el nivel base, con la finalidad de recabar información para que otros miembros del middleware puedan reflexionar sobre él. Este agente supervisa los componentes del nivel base, sus variables de estado y desempeño, entre otras cosas, para

[7] Los procesos de negocios son el conjunto de tareas, reglas y actividades que se ejecutan en un sistema para alcanzar un objetivo específico [23].

permitir posteriormente la reflexión del middleware. El Monitor funciona utilizando un sistema de "polling", recogiendo los valores de las variables a supervisar cada cierto intervalo de tiempo.

Reflector: Este agente procesa y analiza la información del nivel base para tomar decisiones sobre qué ajustes hacerle al mismo. Su función más importante es procesar las variables del ambiente (por ejemplo, los procesos a ejecutar, la capacidad de memoria y de almacenamiento disponible en la plataforma, los algoritmos de cifrado de datos existentes, etc.), con el fin de generar los cambios necesarios (intercesión) en el nivel base.

Manejador de Información: Se trata de un agente que escribe y lee los archivos XML que almacenan la información relacionada con los eventos del sistema base. También tiene mecanismos de aprendizaje que permiten ir actualizando la información/conocimiento almacenado en dichos archivos. Así, de manera general administra la información relacionada a los estados del sistema, a las variables que se monitorean, etc.

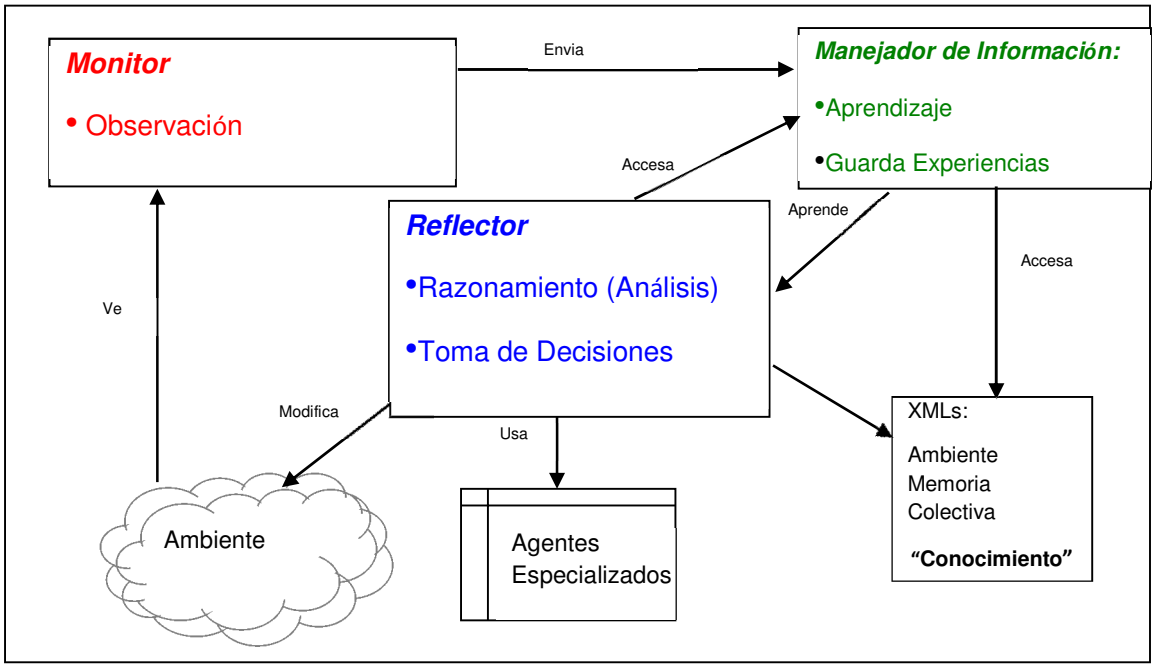

Figura 10: Archivo de configuración inicial – Inicial.xml.

El middleware reflexivo incorpora las siguientes características al nivel base:

- *Auto-conciencia:* El sistema está consciente de sí mismo y tiene presente sus estados y comportamientos.
- *Auto-Eficiencia:* Detecta degradaciones del sistema, e inteligentemente realiza cambios para evitar situaciones peligrosas.
- *Planes Preventivos:* Detecta problemas potenciales, y reconfigura el sistema para mantenerlo en funcionamiento. Esto le da un comportamiento proactivo.
- *Contextualmente Consciente:* Es consciente del ambiente en el cual el sistema base está siendo ejecutado.

El comportamiento de la arquitectura (middleware) que se diseñó; se puede resumir como sigue: El agente monitor observa el comportamiento de la aplicación base, comparando los estados del sistema con los estados esperados; mientras que el agente reflector toma decisiones acerca de cómo mejorar algunas actividades específicas de la aplicación base, estas decisiones están basadas en la información que el agente monitor recoge a través del monitoreo de los estados del sistema base. El agente manejador de información almacena los estados del sistema en archivos XML, dichos estados provienen del agente monitor; se diseñó de esta manera para especializar a los agentes en actividades específicas. En este caso, el manejador de información solo manipula información proveniente de otros agentes, no importa cual agente sea; también posee mecanismos de aprendizaje que le permiten mantener actualizada dicha información. Así mismo, el agente monitor solo está programado para monitorear la aplicación base. Esa información puede ser accedida por el resto de los agentes, a través de los archivos XML, cuando la misma es requerida para tomar una decisión. Así, los archivos XML fungen como vía de comunicación indirecta entre los agentes del middleware (véase fig. 10). Específicamente, los agentes utilizan dos tipos de comunicación: una comunicación basada en mensajes directos, en los casos de la comunicación entre el agente Monitor y Manejador de Información (para enviar los estados, y variables monitoreadas), entre el agente reflector con los especializados (para pedirles apoyo en sus procesos de toma de decisión), o entre el agente Manejador de Información y el Reflector (para aprender a partir de sus decisiones); y una comunicación indirecta establecida utilizando archivos XML. Agentes especializados pueden ser incorporados al middleware, para introducirle nuevas funcionalidades. Normalmente, eso ocurre cuando se desea realizar una implementación particular en un área específica.

Por ejemplo, en este trabajo se incorporó un agente que maneja todos los aspectos relacionados con seguridad, específicamente mecanismos criptográficos, a fin de instanciar el middleware reflexivo para gestionar el tema de seguridad en las aplicaciones en el nivel base (ver capítulo 5). Igualmente, fue instanciado el middleware reflexivo para gestionar el tema de rendimiento; en este caso se incorporaron agentes que calculan el uso de memoria, disco duro y CPU, para cada componente del nivel base (ver capítulo 5).

4.3 Componentes del Middleware:

Un middleware ha sido construido para trabajar como otra capa dentro de la arquitectura de cualquier sistema compuesto por componentes de software. Esto provee la posibilidad de monitorear el ambiente donde la aplicación está implementada, para determinar cuáles componentes del sistema no están funcionando debidamente (según el criterio objetivo del middleware reflexivo), cuáles deben ser remplazados, entre otras cosas. Así, por ejemplo, al implementar el middleware se definen cuáles serán los estados deseados del sistema base, y qué tipo de comportamiento analizará el Monitor, a fin de suministrar la información necesaria para que se puedan tomar las decisiones sobre si modificar o no la aplicación base.

Como se indicó en la sección anterior, los componentes del middleware son tres agentes genéricos, más los agentes especializados para los casos de estudios específicos donde vaya a ser usado el middleware. Para su descripción, usaremos algunos de los modelos y diagramas de la metodología de especificación de agentes llamada MASINA [8].

4.3.1. Agente Monitor

La figura 11 presenta el caso de uso para el agente monitor.

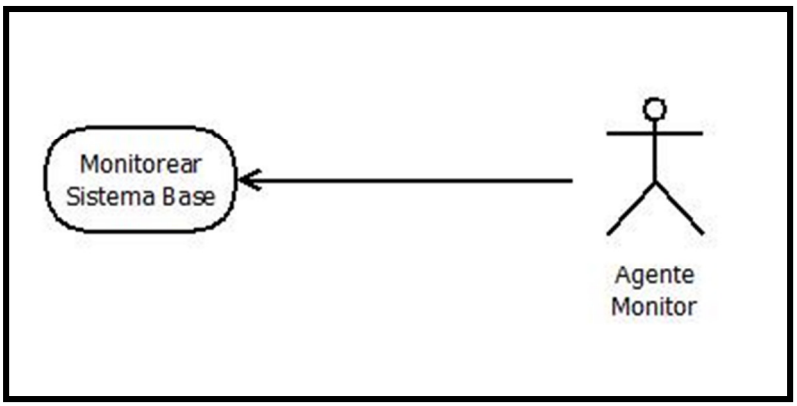

Figura 11: Caso de uso del Agente Monitor

El diagrama de actividades del agente monitor es mostrado en la figura 12.

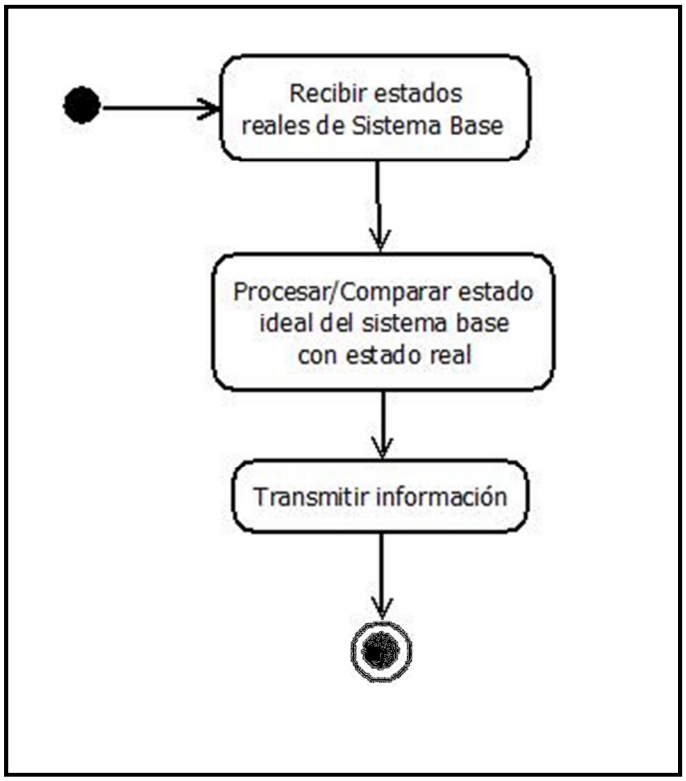

Figura 12: Diagrama de actividades del agente Monitor

Las tablas 1, 2 y 3 describen el modelo de agente, la relación entre los servicios y tareas de dicho agente, así como el modelo de tareas del agente monitor.

Tabla 1: Modelo de Agente para el Agente Monitor

AGENTE MONITOR	
Nombre	Monitor
Posición	Nive Meta del Middleware reflexivo
Componentes	Depende de la instanciación del middleware reflexivo
Descripción	Se encarga de monitorear la aplicación base en tiempo de ejecución, utilizando como entrada archivos de configuración xml. Este archivo contiene los valores aceptables de diferentes estados del sistema base.
OBJETIVO DEL AGENTE MONITOR	
Nombre	Monitoreo
Descripción	Se encarga de comparar los estados ideales de la aplicación base con los estados reales a fin de enviar información relevante al Agente Reflector a través del uso del Agente de Manejo de Información.
Parámetros de entrada	Variables de la aplicación base a monitorear.
Parámetros de salida	Valores de las variables monitoreadas.
Condición de activación	Solicitud de envío/procesamiento de datos
Condición de finalización	Envío/procesamiento de Información
Condición de éxito	Información enviada/procesada
Condición de fracaso	Error en el envío/procesamiento de Información
SERVICIO DEL AGENTE MONITOR	
Nombre	Monitorear la aplicación base.

Descripción	Se encarga de la transmisión de información y datos recolectados al agentes Reflector para su procesamiento, vía un archivo XML gestionado por el agente manejador de información
Parámetros de entrada	Variables a monitorear
Parámetros de salida	Valores recogidos de las variables monitoreadas
SERVICIO DEL AGENTE MONITOR	
Nombre	Comparar/Procesar Información.
Descripción	Se encarga de comparar la configuración ideal del sistema base con el estado real del mismo en un momento dado.
Parámetros de entrada	Valores de las variables monitoreadas
Parámetros de salida	Comparación hecha

Tabla 2: Relación Servicios-Tareas del Agente Monitor

Servicios	Tareas
Monitorear aplicación base	T1. Verificar Estado de la aplicación base. T2. Transmitir Información al agente Manejador de información.
Comparar/Procesar Información	T1. Procesar/acondicionar datos

Tabla 3: Modelo de Tareas del Agente Monitor

TAREA: Verificar Estado de la aplicación base.	
Nombre	Verificar Estado de la aplicación base.
Objetivo	Obtener datos del estado del sistema base.
Descripción	Los datos obtenidos del monitoreo permanente del sistema base se procesan comparándolos con los datos relacionados a los estados ideales requeridos.

Precondición	Existencia de datos y de comunicación
Subtareas	Verificar estado de aplicación base y transmitir información a agente manejador de información.
INGREDIENTE-TAREA Verificar Estado de la aplicación base.	
Nombre	**Contenido**
Tipo de solicitud	Solicitud
Datos	Datos correspondientes al estado del sistema base
ID	Identificador del estado (Fecha y hora de registro de datos)
TAREA: Transmitir Información al agente Manejador de información	
Nombre	Transmitir Información
Objetivo	Transmisión de información y datos recolectados de los procesos y otros agentes del SMA.
Descripción	Los datos generados de los procesos y la información proporcionada por los agentes es recolectada y transmitida para su procesamiento.
Precondición	Debe existir comunicación al menos entre el Agente Monitor y el Agente Manejador de Información.
Subtareas	Ninguna
INGREDIENTE- Transmitir Información al agente Manejador de información	
Nombre	**Contenido**
Tipo de solicitud	Envío
Datos	Información relacionada a las variables de estado de la aplicación base
ID	Identificador del estado (Fecha y hora de registro de datos)
TAREA: Procesar/acondicionar datos	
Nombre	**Contenido**
Objetivo	Procesar/acondicionar datos
Descripción	Los datos, antes de su envío al agente manejador de información, son limpiados, etc.

Precondición	Capacidad de depurar los datos
Subtareas	Ninguna
INGREDIENTE-TAREA Procesar/acondicionar datos	
Nombre	**Contenido**
Tipo de solicitud	Procesamiento
Datos	Datos de estado de la aplicación base
ID	Identificador del estado (Fecha y hora de registro de datos)

4.3.2. Agente Reflector

La figura 13 presenta el caso de uso para el agente Reflector

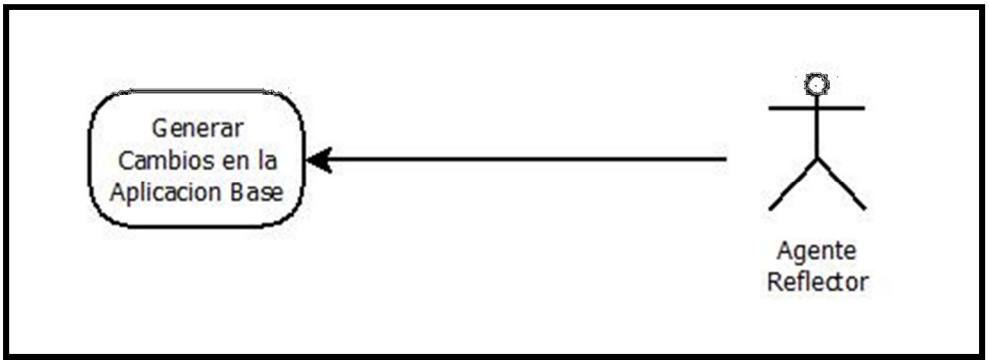

Figura 13: Casos de uso del Agente Reflector

El diagrama de actividades del agente Reflector es mostrado en la figura 14.

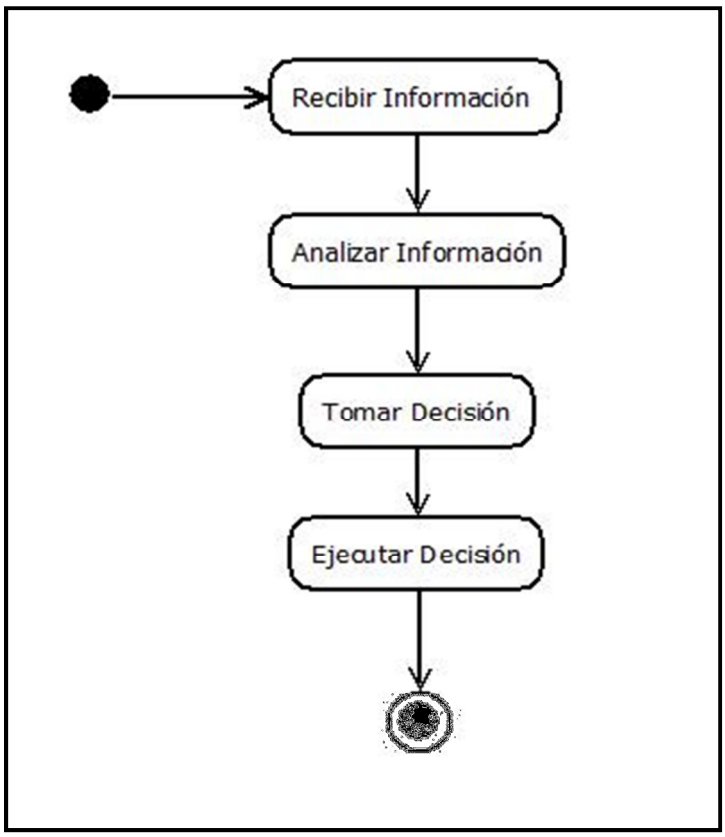

Figura 14: Diagrama de actividades del agente Reflector

Las tablas 4, 5 y 6 a continuación describen el modelo de agente, la relación entre los servicios y tareas de dicho agente, así como el modelo de tareas del agente Reflector.

Tabla 4: Modelo de Agente - Agente Reflector

AGENTE REFLECTOR	
Nombre	Reflector
Posición	Nivel Meta del Middleware reflexivo
Componentes	Depende de la instanciación del middleware reflexivo
Descripción	Se encarga de tomar decisiones sobre la aplicación base, a fin de mantener la misma bajo los parámetros especificados en los archivos iniciales de configuración.
OBJETIVO DEL AGENTE REFLECTOR	
Nombre	Reflexionar/Tomar Decisiones
Descripción	Se encarga de analizar la información de la aplicación base y tomar decisiones a fin de mantenerla bajo los parámetros permitidos.
Parámetros de entrada	Datos de los Procesos de la aplicación base.
Parámetros de salida	Información sobre los procesos de la aplicación base.
Condición de activación	Solicitud de envío/procesamiento de datos
Condición de finalización	Envío/procesamiento de Información
Condición de éxito	Información enviada/procesada
Condición de fracaso	Error en el envío/procesamiento de Información
SERVICIO DEL AGENTE REFLECTOR	
Nombre	Reflexionar sobre la aplicación base.
Descripción	Se encarga de analizar la información de la aplicación base y tomar decisiones a fin de mantenerla bajo los parámetros permitidos.
Parámetros de entrada	Variables de estado de la aplicación base y variables sobre estados anteriores de la aplicación base.
Parámetros de salida	Información

Tabla 5: Relación Servicio - Tareas del Agente Reflector

Servicios	Tareas
Reflexionar sobre la aplicación base	T1. Analizar la información disponible. T2. Llamar agente especializado. T3. Enviar Alarma. T4. Buscar componente. T5. Hacer cambios al nivel base T6. Transmitir Información al agente Manejador de Información.

Tabla 6: Modelo de Tareas del Agente Reflector

TAREA: Analiza información	
Nombre	Analiza
Objetivo	Decidir la mejor acción para hacer más eficiente el sistema base.
Descripción	Se analiza toda la información disponible y se toma la mejor acción para hacer mas eficiente el sistema base.
Servicios asociados	Gestionar información
Precondición	Existencia de datos y de comunicación
Subtareas	Analizar la información disponible.
INGREDIENTE- Analiza información	
Nombre	**Contenido**
Tipo de solicitud	Solicitud
Datos	Datos correspondientes al estado del sistema base

ID	Identificador del estado (Fecha y hora de registro de datos)
TAREA: TOMAR DECISION	
TAREA: Llamar agente especializado	
Nombre	Llama Agente Especializado
Objetivo	Decidir a qué agente especializado enviar la solicitud de ejecución de una actividad específica requerida.
Descripción	En base a la información analizada, se llama al agente especializado respectivo, a fin de ejecutar la acción necesaria que permita mejorar los estados del sistema base según lo establecido en los requerimientos iníciales.
Servicios asociados	Ejecución de acciones especiales
Precondición	Existencia de datos y de comunicación
Subtareas	Acciones especializadas.
INGREDIENTE- Llamar agente especializado	
Nombre	**Contenido**
Tipo de solicitud	Solicitud
Datos	Datos relacionados a la acción especial que se requiere.
ID	Identificador del estado (Fecha y hora de registro de datos) así como el nombre de la tarea ejecutada.
TAREA: Enviar Alarma	
Nombre	Alarma
Objetivo	Generar una alarma que permita alertar al sistema y a los administradores sobre los cambios necesarios a realizar.
Descripción	Se genera una alarma que permite alertar sobre los cambios realizados o necesarios por ejecutar.

Servicios asociados	Gestionar información
Precondición	Existencia de datos y de comunicación
Subtareas	Analizar la información disponible.
INGREDIENTE- Enviar Alarma	
Nombre	**Contenido**
Tipo de solicitud	Envío
Datos	Información sobre acción a ejecutar.
ID	Identificador del estado (Fecha y hora de registro de datos) nombre de la acción.
TAREA: Buscar Componente	
Nombre	Búsqueda.
Objetivo	Buscar y seleccionar el mejor componente disponible a fin de reemplazar aquel que no cumple con los requerimientos del sistema base.
Descripción	Se ejecuta el algoritmo de selección detallado en el capítulo 3, a fin de seleccionar el mejor componente que pueda ser utilizado como reemplazo del componente que no cumple con los requerimientos mínimos de funcionamiento de la aplicación base.
Servicios asociados	Búsqueda de Componentes
Precondición	Existencia de algoritmo de selección de componentes
Subtareas	Reemplazo de componente.
INGREDIENTE- Buscar Componente	
Nombre	**Contenido**
Tipo de solicitud	Acción.
Datos	Datos correspondientes al componente que se desea reemplazar.

ID	Identificador del estado (Fecha y hora de registro de datos) – Componente a reemplazar.
TAREA: Cambios al Nivel Base	
Nombre	Modifica – Intercesión.
Objetivo	Modificar componente de la aplicación base.
Descripción	Se hacen cambios a los componentes de la aplicación base, pueden ser a nivel de configuración, tiempo de ejecución, modo de ejecución componentes usados, etc.
Servicios asociados	Depende de cada cambio
Precondición	Permiso de escritura y lectura en los componentes de la aplicación base.
Subtareas	Ninguna
INGREDIENTE- Cambios al Nivel Base	
Nombre	**Contenido**
Tipo de solicitud	Acción
Datos	Datos correspondientes al estado del sistema base en un momento particular de su ejecución y datos relacionados con los mismos estados en momentos anteriores que permiten hacer comparaciones de estado.
ID	Identificador del estado (Fecha y hora de registro de datos) – Cambio realizado – Nombre del componente
TAREA: Transmitir información	
Nombre	Envía Información
Objetivo	Enviar información para ser almacenada
Descripción	Se envía toda la información relacionada con la decisión tomada y la acción ejecutada al agente manejador de información, a fin de ser almacenada.
Servicios asociados	Gestionar información

Precondición	Existencia de datos y de comunicación
Subtareas	Analizar la información disponible.
INGREDIENTE- Transmitir información	
Nombre	**Contenido**
Tipo de solicitud	Envía
Datos	Datos correspondientes a la acción ejecutada
ID	Identificador del estado (Fecha y hora de registro de datos) – acción ejecutada

4.3.3. Agente manejador de información

La figura 15 presenta el caso de uso para el agente manejador de información (info).

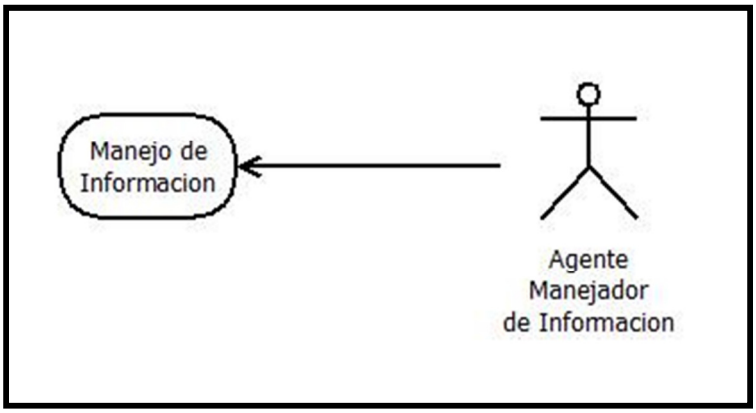

Figura 15: Casos de uso del Agente Manejador de Información

La figura 16 presenta el diagrama de actividades del agente Manejador de Información.

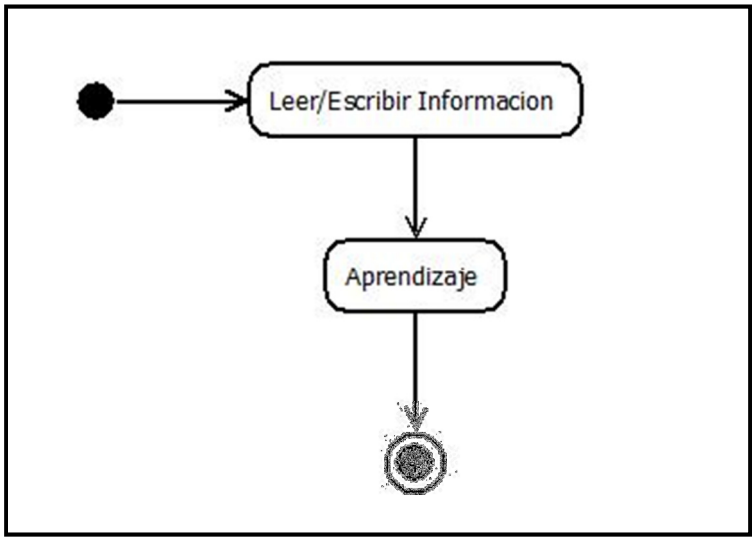

Figura 16: Diagrama de actividades del agente Manejador de Información

Tabla 7: Modelo de Agente para Agente Manejador de Información

AGENTE MANEJADOR DE INFORMACION	
Nombre	Info
Posición	Nivel Meta del Middleware reflexivo
Componentes	Depende de la instanciación del middleware reflexivo
Descripción	Se encarga de leer y escribir datos en los archivos XML, a fin de asegurar la comunicación indirecta entre los agentes.

OBJETIVO DEL AGENTE MANEJADOR DE INFORMACION	
Nombre	Manejar archivos de configuración y comunicación.
Descripción	Se encarga de leer y escribir información en los archivos necesarios para garantizar la comunicación indirecta entre los agentes.
Parámetros de entrada	Datos de los Procesos de la aplicación base.
Parámetros de salida	Información de los procesos de la aplicación base.
Condición de activación	Solicitud de envío/procesamiento de datos
Condición de finalización	Envío/procesamiento de Información
Condición de éxito	Información enviada/procesada
Condición de fracaso	Error en el envío/procesamiento de Información
SERVICIO DEL AGENTE MANEJADOR DE INFORMACION	
Nombre	Administrar los archivos XML.
Descripción	Lectura y escritura de archivos XML.
Parámetros de entrada	Solicitud de almacenamiento de Información
Parámetros de salida	Información
SERVICIO DEL AGENTE INFO	
Nombre	Aprender
Descripción	Aprender desde las fuentes de información disponibles.
Parámetros de entrada	Solicitud de almacenamiento de Información
Parámetros de salida	Información

Tabla 8: Relación Servicios - Tareas del Agente Manejador de Información

Servicios	Tareas
Administrar los archivos XML sobre la aplicación base	T1. Escribe Información. T2. Obtiene/Lee información.
Aprender	T1. Actualiza información.

Tabla 9: Modelo de Tareas del Agente Manejador de Información

TAREA: Obtiene/Lee información	
Nombre	Obtiene/Lee información
Objetivo	Leer información almacenada en los archivos XML.
Descripción	Leer archivos XML contentivos de información relacionada a la aplicación base.
Precondición	Existencia de datos y de comunicación, permisos de lectura y escritura a archivos en disco.
Subtareas	Lee/escribe información.
INGREDIENTE-TAREA Obtiene/Lee información	
Nombre	**Contenido**
Tipo de solicitud	Solicitud
Datos	Información relacionada con la aplicación base, su configuración y cambios.
ID	Identificador del estado (Fecha y hora de registro de datos) – acción ejecutada
TAREA: Escribe Información.	
Nombre	Escribe Información.
Objetivo	Escribe y lee información en archivos XML.
Descripción	Escribe archivos XML contentivos de información relacionada a la aplicación base.
Precondición	Existencia de datos y de comunicación, permisos de lectura y escritura a archivos en disco.
Subtareas	Ninguna
INGREDIENTE-TAREA Escribe Información	
Nombre	**Contenido**
Tipo de solicitud	Acción
Datos	Información relacionada con la aplicación base, su configuración y cambios.
ID	Identificador del estado (Fecha y hora de registro de datos) – acción ejecutada

TAREA: Actualiza información.	
Nombre	Escribe
Objetivo	Actualizar información a partir del resultado del proceso de aprendizaje.
Descripción	Realiza el proceso de aprendizaje a partir del quehacer en el middleware y su gestión de la aplicación base, y después actualiza la información resultante del mismo.
Precondición	Permisos de lectura y escritura, disponibilidad de la información a almacenar.
Subtareas	Ninguna
INGREDIENTE-TAREA Actualiza información	
Nombre	**Contenido**
Tipo de solicitud	Acción
Datos	Información relacionada con la aplicación base, su configuración y cambios aprendidos.
ID	Identificador del estado (Fecha y hora de registro de datos) – acción ejecutada

4.3.4. Agente Especializado

Los agentes especializados dependen del caso de uso donde será usado el middleware reflexivo. Los mismos realizan tareas específicas requeridas por el middleware para sus procesos de reflexión, que le permitirán realizar la toma de decisiones que conllevan a los cambios requeridos en las aplicaciones en el nivel base, para mantener los criterios objetivos que se persiguen con el middleware. En este trabajo se hizo la implementación del middleware para dos casos específicos, uno relacionado con seguridad de la aplicación base, y otro con el rendimiento de la misma. Aquí presentamos el esquema general de un agente especializado

La figura17 representa el caso de uso general para un agente especializado

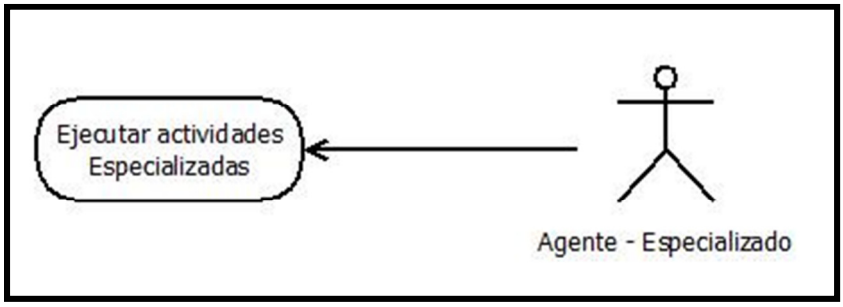

Figura 17: Casos de uso del agente especializado

El diagrama de actividades del agente especializado es mostrado en la figura 18.

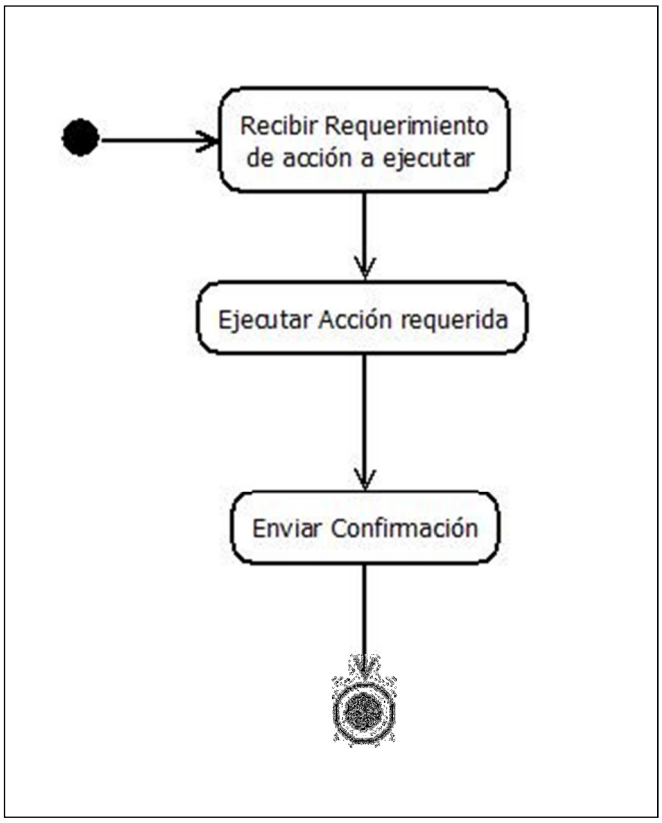

Figura 18: Diagrama de actividades del agente especializado

Las tablas 10, 11 y 12 describen el modelo de agente, la relación entre los servicios y tareas de dicho agente, así como el modelo de tareas del agente especializado.

Tabla 10: Modelo de Agente para el Agente Especializado

AGENTE ESPECIALIZADO	
Nombre	Agente Especializado
Posición	Nivel Meta del Middleware reflexivo
Componentes	Depende de la instanciación
Marco de Referencia	Depende de la instanciación
Descripción	Se encarga de ejecutar las tareas especializadas que no pueden ser manejadas por los agentes originales del middleware, cuando deben hacer un procesamiento particular con la información proveniente de la aplicación base.
OBJETIVO DEL AGENTE ESPECIALIZADO	
Nombre	Actividades Especiales
Descripción	Se encarga de ejecutar acciones que el middleware no logra hacer con los agentes básicos que posee. Dichas acciones son específicas a requerimientos particulares que se deseen implementar en la aplicación base.
Parámetros de entrada	Acción a ejecutar.
Parámetros de salida	Confirmación de ejecución.
Condición de activación	Solicitud de envío
Condición de finalización	Envío
Condición de éxito	Acción ejecutada
Condición de fracaso	Error en la ejecución
SERVICIO DEL AGENTE ESPECIALIZADO	
Nombre	Actividad especializada
Descripción	Se encarga de la ejecución de las acciones especializadas, por ejemplo cambio de mecanismo de encriptamiento, liberación de memoria etc.

Parámetros de entrada	Acción a ejecutar
Parámetros de salida	Confirmación

Tabla 11: Relación Servicios-Tareas del Agente Especializado

Servicios	Tareas
Actividad especializada	T1.Ejecutar acciones especializadas relacionadas a la aplicación base.

Tabla 12: Modelo de Tareas del Agente Especializado

TAREA: Ejecutar acciones especializadas relacionadas a la aplicación base.	
Nombre	Ejecutar acciones especializadas relacionadas a la aplicación base
Objetivo	Ejecutar procesos especializados referido a la gestión de la aplicación base
Descripción	Se ejecuta la acción solicitada por el Agente Reflector
Precondición	Existencia de datos y de comunicación, y la capacidad para realizar la tarea encomendada
Subtareas	Enviar Información al agente reflector
INGREDIENTE-TAREA Ejecutar acciones especializadas relacionadas a la aplicación base.	
Nombre	**Contenido**
Tipo de solicitud	Solicitud
Datos	Datos necesarios para realizar la tarea especializada. Por ejemplo cambio de mecanismo de encriptamiento, liberación de memoria etc.
ID	Identificador del estado (Fecha y hora de registro) de los datos

4.3.5. Modelos de Coordinación y Comunicación del Sistema

Las conversaciones describen los diferentes momentos comunicantes en la comunidad de agentes del middleware, para la realización de tareas colectivas. El middleware ejecuta dos conversaciones generales para mantener una aplicación base bajo parámetros y condiciones preestablecidas:

- Supervisar una aplicación base, integra los siguientes actos de habla
- Mantener una aplicación base bajo ciertas condiciones preestablecidas

Pasemos a describir cada conversación, para ello usaremos las plantillas de MASINA para definir las conversaciones y actos de hablas presentes en un Sistema Multiagentes, y el diagrama de interacción de UML.

4.3.6. Supervisar una aplicación base, integra los siguientes actos de habla:

Tabla 13: Conversaciones entre agentes para supervisar aplicación base

CONVERSACIÓN: Supervisar una aplicación base	
Objetivo	Vigilar la aplicación base, estimar los valores aceptables de estado del sistema base, y mantener actualizado los archivos XML respectivos para poder actuar en caso que se requiera hacer un cambio.
Agentes	Agente Monitor, Agente Info, archivos XML
Iniciador	Agente Monitor
Actos de Habla	Enviar solicitud información, Regresa Info, Almacena Información en XML
Precondición	Existir una solicitud de supervisión a una aplicación base
Condición de Terminación	permanente
Descripción	El agente Monitor supervisa permanentemente la aplicación base tomando en cuenta los valores esperados y los valores reales de los diferentes estados de la misma. Estos valores son almacenados en archivos XML, estos últimos son manipulados por el agente Info.

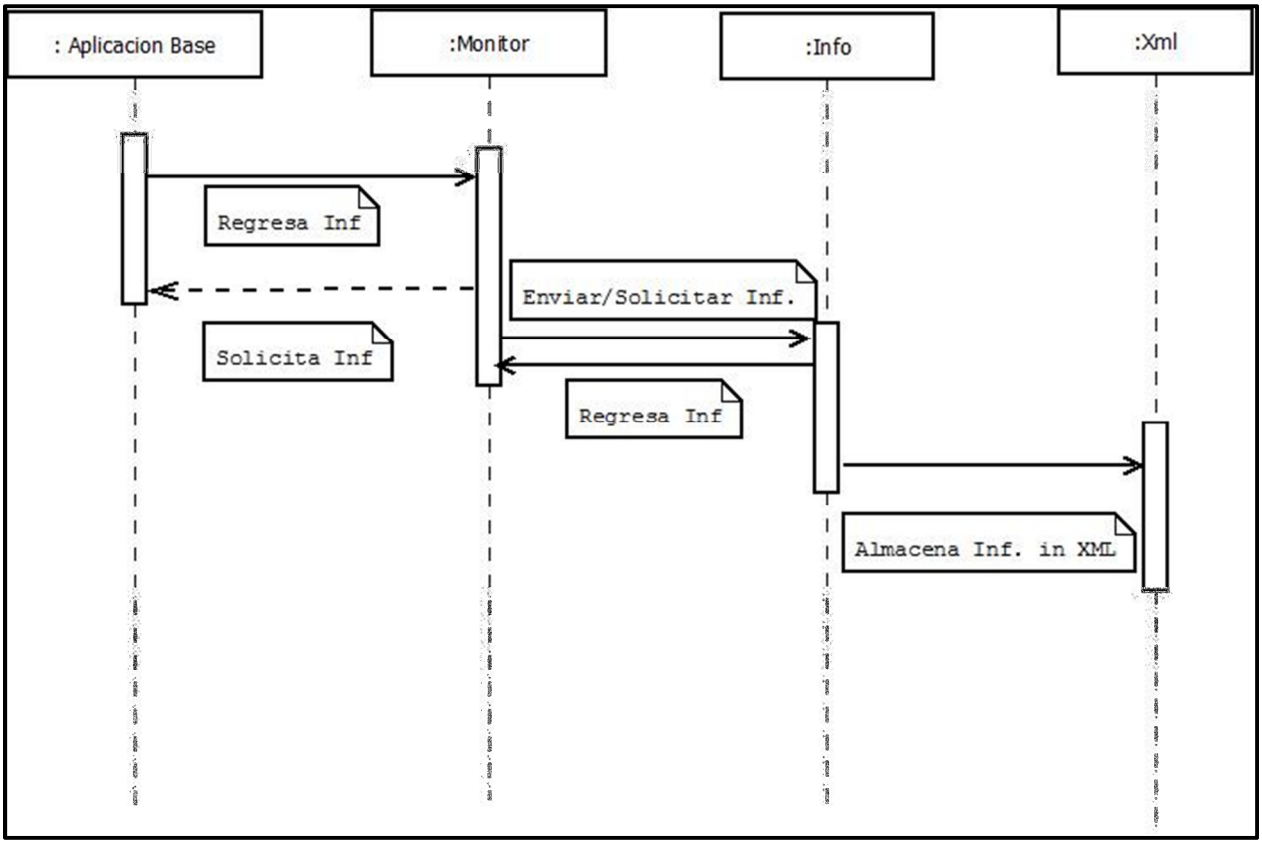

Figura 19: Diagrama de Interacción de la Conversación Supervisar una aplicación base

Los actos de habla presentes en esta conversación son:

Tabla 14: Actos de habla entre agentes del middleware para supervisar aplicación base

NOMBRE	Enviar/Solicitar información
Tipo	Informar/solicitar
Objetivo	Transmitir Información relacionada a la aplicación base, o solicitarle información al agente info
Agentes	Agente Monitor, Agente Info
Iniciador	Agente Monitor
Precondición	Existencia de información
Condición de Terminación	Recepción de información

Conversaciones	Supervisar una aplicación base
Descripción	El agente Monitor envía información relacionada con la aplicación base, para hacer la comparación de estados entre los valores esperados y los valores reales. También puede solicitarle una información en específico al Agente Manejador de Información.
NOMBRE	**Regresa Información**
Tipo	Informar
Objetivo	Retornar información. requerida por el Agente Monitor,
Agentes	Agente Monitor, y Agente Manejador de Información o Aplicación Base
Iniciador	Agente Manejador de Información o Aplicación Base
Precondición	Existencia de información.
Condición de Terminación	Recepción de información.
Conversaciones	Supervisar una aplicación base
Descripción	El agente Manejador de Información o la o Aplicación Base retornan una información al agente monitor, previamente solicitada por él.
NOMBRE	**Almacena Información en XML**
Tipo	Almacenar
Objetivo	Almacenar información en archivos XML
Agentes	Agente Manejador de Información, Archivo XML
Iniciador	Agente Manejador de Información
Precondición	Existir información a almacenar
Condición de Terminación	Finalización del almacenamiento de la información
Conversaciones	Supervisar una aplicación base
Descripción	El agente Manejador de Información escribe la información pertinente en archivos XML, que luego pueden ser accedidos por otros agentes para verificar estados anteriores y acciones ejecutadas en el sistema base

4.3.7. Conversación – Mantener una aplicación base bajo ciertas condiciones preestablecidas:

Tabla 15: Conversación entre agentes del middleware para supervisar aplicación base

CONVERSACIÓN: Mantener una aplicación base bajo ciertas condiciones preestablecidas	
Objetivo	Ejecutar acciones sobre la aplicación base, para mantenerla funcionando de acuerdo a los criterios de funcionamiento definidos en el middleware
Agentes	Agente Reflector, Agente Especializado, Agente Manejador de Información
Iniciador	Agente Reflector,
Actos de Habla	Solicitar Acción. Ejecutar Cambio, enviar información relacionada a decisión, escribe, lee. Lee información. Confirma Acción
Precondición	Existir una aplicación base sobre el cual hacer los análisis respectivos.
Condición de Terminación	Cuando se haya estabilizado la aplicación base para los criterios de funcionamiento establecidos para el middleware
Descripción	El agente Reflector toma decisión sobre qué acción ejecutar sobre la aplicación base, y la ejecuta para mantenerla en los parámetros deseados. . Para ello se ayuda de otros agentes (agentes especializados, agente info, etc.)

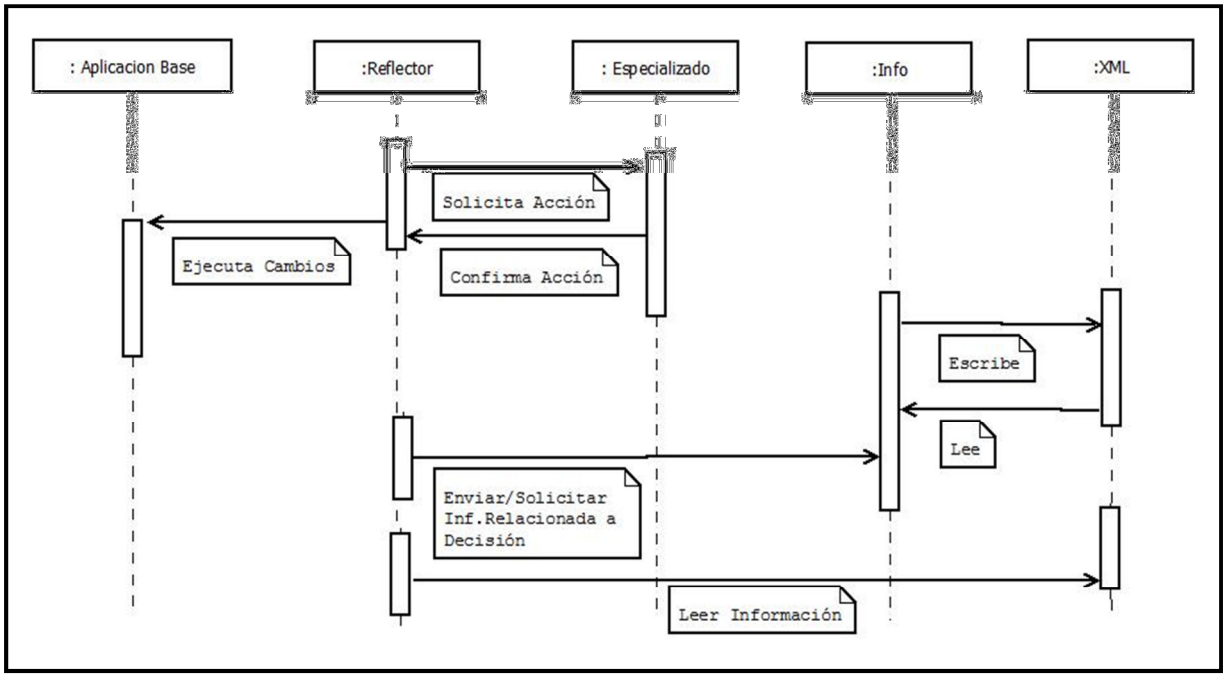

Figura 20: Diagrama de Interacción de la Conversación para mantener una aplicación base bajo ciertas condiciones preestablecidas

Los actos de habla presentes en esta conversación son:

Tabla 16: Actos de habla entre agentes del middleware para mantener aplicación base bajo condiciones preestablecidas

NOMBRE	Solicitar Acción
Tipo	Solicitar
Objetivo	Solicitar una acción específica a un agente especializado
Agentes	Agente especializado, Agente Reflector
Iniciador	Agente Reflector
Precondición	Permiso de lectura, escritura y ejecución.
Condición de Terminación	Acción realizada.

Conversaciones	Mantener una aplicación base bajo ciertas condiciones preestablecidas
Descripción	El Agente Reflector solicita algún tipo de análisis especializado sobre el comportamiento de la aplicación base, dicho análisis es realizado por uno de los agentes especializados del middleware.
NOMBRE	Confirmar acción
Tipo	Informar
Objetivo	Confirmación de la acción ejecutada por parte del agente especializado, que puede implicar regresar algún resultado.
Agentes	Agente Reflector – Agente Especializado
Iniciador	Agente Especializado
Precondición	Requerimiento de acción.
Condición de Terminación	El Agente Especializado manda estatus de terminación de tarea solicitada.
Conversaciones	Mantener una aplicación base bajo ciertas condiciones preestablecidas
Descripción	El Agente Especializado ejecuta la acción requerida por el agente Reflector, cuando la misma está relacionada con el caso de uso, es decir cambio de mecanismo de encriptamiento, liberación de memoria entre otras; además envía mensaje de confirmación de la ejecución (quizás con los resultados).
NOMBRE	enviar información relacionada a decisión
Tipo	Informar
Objetivo	Enviar información a Agente Manejador de Información para mantener registro de decisiones en los archivos XML
Agentes	Reflector – Manejador de Información
Iniciador	Reflector
Precondición	Existencia de decisión
Condición de Terminación	Finaliza ejecución de la decisión.

Conversaciones	Mantener una aplicación base bajo ciertas condiciones preestablecidas
Descripción	El agente Reflector envía información relacionada con la decisión tomada, con la finalidad de que la misma quede almacenada y pueda ser leída posteriormente
NOMBRE	escribe/lee.
Tipo	Envía/recibe
Objetivo	Enviar y recibir información al archivo XML
Agentes	Manejador de Información
Iniciador	Manejador de Información
Precondición	Existencia de información
Condición de Terminación	Finaliza la escritura/lectura de archivo XML
Conversaciones	Mantener una aplicación base bajo ciertas condiciones preestablecidas
Descripción	El agente Info lee/escribe los archivos XML con información relacionada con los estados de la aplicación base. De esta manera, mantiene actualizado dichos archivos
NOMBRE	Lee información
Tipo	Leer
Objetivo	Buscar información de su interés para el proceso de toma de decisiones en los archivos XML
Agentes	Reflector
Iniciador	Reflector
Precondición	Existencia de archivo XML y acción a ejecutar
Condición de Terminación	Lectura de archivo
Conversaciones	Se lee archivo XML para obtener información.
Descripción	EL Reflector puede acceder a archivos XML que almacenan información histórica, con la finalidad de verificar datos que permitan tomar la mejor decisión posible.
NOMBRE	Ejecutar Cambio.

Tipo	Ejecutar
Objetivo	Modificar la aplicación base según las decisiones tomadas por él para mantenerla funcionando en los parámetros deseados
Agentes	Agente reflector
Iniciador	Agente reflector
Conversaciones	Mantener una aplicación base bajo ciertas condiciones preestablecidas
Descripción	Envía orden a Agente especializado para ejecutar los cambios requeridos.

4.4 Implementación

El middleware fue programado como una biblioteca Java. Los tres agentes principales del middleware son implementados a través de tres clases principales, una de ellas caracteriza al reflector (Reflector), otra al Monitor, y la tercera al Manejador de Información (Info) (ver figura 21). Aparte de estas tres clases, otras clases particulares pueden ser definidas para implementar a los agentes especializados, que dependen según para que se use el middleware (ver capítulo 5 donde se indican, en dos casos de estudios, las respectivas clases especializadas).

La especificación de los métodos es establecida en los modelos de tareas definidos previamente (ver tablas 3, 6, 9 y 12).

El middleware debe ejecutarse como proceso en el mismo ambiente donde la aplicación base está implementada. Es decir, el middleware se utiliza como una aplicación ejecutable que se mantiene corriendo en el mismo ambiente donde la aplicación base se ejecuta. Para usar el sistema se deben realizar los siguientes pasos:

1. Se debe configurar el archivo XML de inicio que almacena los estados deseados del sistema, la aplicación base y los componentes que la conforman, que serán monitoreados por el middleware (ver figura 22). Ese archivo establece el enlace entre el middleware y la aplicación base, ya que en él se indica el nombre de la aplicación base, los procesos que ejecuta, y los componentes que mantiene y que deben ser monitoreados.

2. Se debe copiar el ejecutable del middleware, ya compilado, así como el ejecutable del algoritmo de selección y búsqueda de componentes, y el archivo XML descrito, en el ambiente donde se ejecutarán. Después, se debe iniciar la ejecución de los mismos manualmente, conjuntamente con la aplicación base, y dichos procesos se mantendrán en ejecución hasta que manualmente se paren,en el caso de la aplicación base, cuando ella termine.

3. Ya con lo anterior hecho, el middleware inicia la ejecución de su sistema multiagentes para el monitoreo de la aplicación base. Para ello él realiza las llamadas siguientes :

```
//Creación del agente Monitor
Monitor monitor= newMonitor();
//Creación del agente Info (InformationManager)
InformationManager configuration = newInformationManager();
//Se cargan todas las variables configurables de la librería
//Variable que almacena sistema operativo de la aplicación base
String OS = newString();
// Variable que mantiene el usuario actual de la app. base
String current_user = newString();
//Propiedades particulares a considerar.
//Como los estados del sistema.
Properties current_prop = new       Properties();
String specific_prop = newString();
configuration.set_Security();
// Se inicia el monitoreo –
// Este grupo de funciones se debe dividir
// según lo que se desee que la librería haga,
//ejemplo para el caso de monitoreo del Rendimiento
int processorsNum = Runtime.getRuntime().availableProcessors();
long freeMem = Runtime.getRuntime().freeMemory();
long totalMem = Runtime.getRuntime().totalMemory();
long usedMem = totalMem - freeMem;
OS=System.getProperty("os.name");
// ejemplo para el caso de monitoreo de la Seguridad
current_user = System.getProperty("user.name");
```

```
 Algoritmo=Seguridad.Alg_Encripcion();
//Se crea el agente Reflector
Reflector reflect = newReflector();
//Agente reflector inicia análisis de la información
// según objetivo del middleware
//(se muestra un ejemplo en seguridad)
reflect.analizing_security(current_user);
reflect.decide();
```

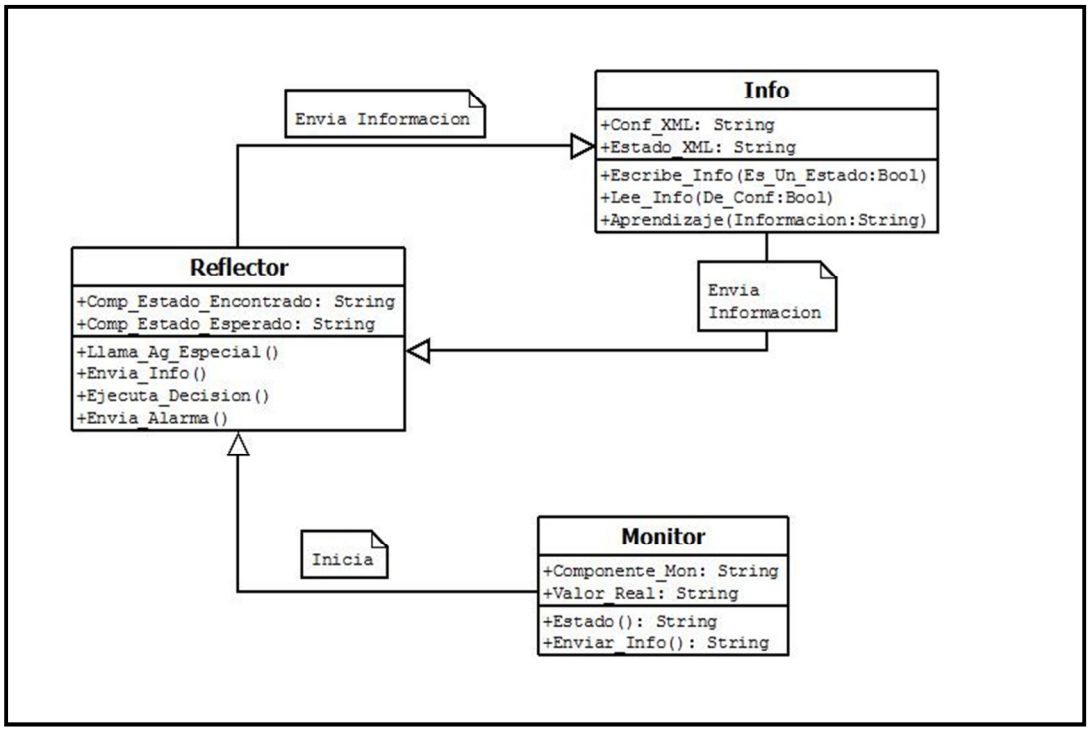

Figura 21: Diagrama de Clases del Middleware

```
<MIDDLEWARE CONFIGURATION>03-03-08</MIDDLEWARE CONFIGURATION>
<Application>
<AppName>Reflective Architecture Test<AppName>
<SoftwareComponent>
      <Name>Geometry</Name>
<Path>c:/componentloc/geom</Path>
      <Security>
            <User>BLANCA ABRAHAM</User>
            <Rights>111</Rights>
            <Levels>
                  <Confidenciality>0</Confidenciality>
                  <Integrity>0</Integrity>
                  <Authentication>0</Authentication>
            </Levels>
      </Security>
```

Figura 22: Archivo de configuración inicial – Inicial.xml

En el capítulo 5 se explica con más detalle el uso del middleware, específicamente para dos casos de uso, y en el anexo 1 se puede revisar el manual de sistema del mismo.

Capítulo 5

Casos de Estudio

En este capítulo se muestran los casos de estudio que fueron implementados a fin de evaluar el funcionamiento del middleware reflexivo, así como los experimentos para evaluar el rendimiento de los dos algoritmos de búsqueda y selección de componentes de software propuestos en esta tesis. También, se hace un análisis de las posibles formas de implementar y utilizar el middleware reflexivo.

5.1 Selección inteligente de componentes de software:

Los algoritmos AA y AB se basan en un algoritmo inspirado en inteligencia artificial colectiva, los cuales incorporan la utilización de una variable asociada a cada componente; la cual es equivalente a la feromona que las hormigas utilizan para marcar el camino más eficiente hacia una fuente de alimento. Así; los agentes que seleccionan estos algoritmos incrementarán o disminuirán la cantidad de feromona de cada componente dependiendo de su rendimiento. De esta manera, se espera que luego de que los algoritmos han sido ejecutados y varios componentes han sido seleccionados, la posibilidad de seleccionar los componentes idóneos se incremente. A fin de probar esta hipótesis, se diseñaron los casos de uso que se explican a continuación, en los cuales se comparan los resultados de los algoritmos de selección AA y AB con sistemas de selección bastante conocidos que son utilizados por repositorios como freshmeat y SourceForge.net.

A diferencia de los algoritmos AA y AB, que hacen búsquedas de componentes de software a partir de las especificaciones funcionales y no funcionales del componente deseado, incluyendo además una variable adicional de feromona según la teoría de inteligencia colectiva, FreshMeat y SourceForge.net son dos repositorios de componentes de software, los cuales hacen las búsquedas basadas en palabras claves y criterios tales como [43, 44]:

a. Popularidad: Mide la frecuencia con que diferentes usuarios usan un componente. Es basado en el número de veces que un componente ha sido utilizado. Mientras más uso tenga un

componente más popular es el mismo, aun cuando el número de descargas no sea muy elevado.

b. Vitalidad: Está basado en el número de veces que un componente se actualiza en un periodo de tiempo. Componentes con muchas actualizaciones, los cuales tienen mucho tiempo en un repositorio, y han sido recientemente actualizados con una nueva versión, poseen un alto score de vitalidad; mientras que componentes viejos, con pocas actualizaciones, obtendrán bajo nivel de vitalidad.
c. Clasificación: este es un criterio usado por FreshMeat, donde cada usuario registrado en FreshMeat puede evaluar un componente. Basado en estas evaluaciones, se construye una lista (clasificación) de los 20 componentes más populares.
d. Número de descargas: Número de veces que un componente ha sido bajado de un repositorio. No toma en consideración si el componente es utilizado o no.
e. Categoría: Este es un criterio usado por SourceForge.net para categorizar los componentes que están activos en SourceForge.net. SourceForge.net aplica una fórmula de categorización a partir de todos los datos estadísticos sobre los componentes (tráfico en la página web del componente, número de desarrolladores alrededor del componente, entre otros). SourceForge.net hace que los componentes que se categorizan como altos, aparezcan en el tope de la lista cuando se está buscando un componente.

Por otro lado, los algoritmos de búsqueda AA y AB inicializan el feromona aleatoriamente como un número positivo entre 0 y 1 para cada componente. Así mismo, se necesita definir los requerimientos iníciales de búsqueda y selección de los componentes, específicamente los aspectos funcionales y no funcionales a fin de que los algoritmos puedan ejecutarse de manera correcta y encontrar componentes que reúnan las cualidades esperadas. En los experimentos que se muestran a continuación, ambos algoritmos se ejecutaron para que buscaran componentes que tuvieran, entre otras cosas, las siguientes características funcionales y no funcionales:

- Funcionen en un tipo de sistema operativo particular.
- Estén hechos en un cierto lenguaje de programación.
- Realicen una funcionalidad específica.

En la tabla 17 se muestran las especificaciones de los diferentes componentes que serán buscados en los

diferentes experimentos que se realizan en esta subsección. Los resultados fueron comparados con los resultados arrojados por SourceForge y FreshMeat [43, 44], cuando se hizo la búsqueda utilizando los mismos criterios.

Tabla 17: Requerimientos que deben cumplir los componentes a buscar

Req. Funcional	Req. No Funcional 1	Req. No Funcional 2	Req. No Funcional 3
Operaciones Matemáticas Básicas	Basado en clases y métodos	Lenguaje de Prog. JAVA	Sistema Operativo Win o Linux
Manejo de Archivos de texto	Basado en clases y métodos	Lenguaje de Prog. JAVA	Sistema Operativo Win o Linux
Convertidor de archivos de diferentes tipos	Basado en clases y métodos	Lenguaje de Prog. JAVA	Sistema Operativo Win o Linux

5.2 Experimentos

Para realizar los experimentos se simularon repositorios de componentes de software cuyos componentes poseen archivos XML que describen sus respectivos perfiles, características y feromona asociada; así mismo, se hicieron las búsquedas en FreshMeat y en SourceForge a fin de comparar los resultados con los obtenidos utilizando los algoritmos propuestos. Se utilizaron como requerimientos iníciales los definidos en la Tabla 17.

Experimento A: *Selección de un componente de software:*

En este caso se hizo la corrida de los algoritmos para buscar solo un componente que debe cumplir con los requerimientos especificados en la tabla 17. Para esta prueba se tomó para búsqueda el componente que realiza operaciones matemáticas básicas.

La tabla 18 muestra los resultados de la búsqueda y selección de un componente de software (el primer componente especificado en la tabla 1-ver fila 1), usando los algoritmos de búsqueda de los repositorios freshmeat y sourceforge, y usando nuestros algoritmos de búsquedas (se listan los 3 componentes encontrados con más alto valor para cada criterio, según [43, 44], en los casos de los componentes

seleccionados por los buscadores de los repositorios, o con más alto valor de probabilidad, en el caso de nuestros algoritmos).

En este caso, AA es bastante rápido pues solo necesita crear e inicializar un agente inteligente, el cual va a buscar al menos en un repositorio. Por otra parte, AB procesa la misma tarea creando n agentes, los cuales buscan en diferentes repositorios, pudiendo seleccionar los mismos tipos de componentes. AB genera más opciones de posibles componentes que funcionan bien en el sistema que AA, pues tiene más agentes buscando diferentes soluciones. Solo el agente 3 de AB consigue los mismos componente que AA (ver última fila de la tabla 18).

La popularidad es una característica que depende de las veces que un componente ha sido usado [43, 44], pero no siempre es cierto que los componentes más usados (los componentes más populares), serán los ideales para un sistema en particular. La vitalidad [43, 44], por otra parte, depende de las actualizaciones, que no siempre se relacionan con un mayor rendimiento o mejores cualidades técnicas. Un componente puede tener una alta vitalidad, popularidad y clasificación, y aún así no tener un buen rendimiento. AA y AB usan no solo los requerimientos iníciales (funcionales y no funcionales), sino que también actualizan un valor de feromona que se incorpora en el archivo XML asociado al componente, para un dominio en particular, que caracteriza el comportamiento de ese componente en ese dominio. Otro aspecto interesante del algoritmo B es que puede buscar componentes en más de un repositorio a la vez (ver los resultados devueltos por sus agentes en la tabla 18).

Tabla 18: Selección de un componente

Criterio o Algoritmo Usado	Mejores Componentes
Popularidad – Encontrado en FreshMeat	Componente N. 1, 2, 3
Vitalidad – Encontrado enFreshMeat	Componente N. 3, 15, 3
Vitalidad – Encontrado en SourceForge	Componente N. 12, 21, 11
Clasificación - Encontrado enFreshMeat	Componente N. 17, 18, 1
Categoría – Encontrado en SourceForge	Componente N. 22, 25, 12
Algoritmo A – Encontrados en FreshMeat	Componente N. 3, 4, 6

Algoritmo B – Agente 1 Encontrados en Freshmeat Encontrado en SourceForge	Componente N. 3, 4, 10
Algoritmo B – Agente 2 Encontrado en SourceForge	Componente N. 12, 14, 15
Algoritmo B – Agente 3 Encontrados en FreshMeat	Componente N. 3, 4, 6

Algo interesante a resaltar en este primer experimento es que los componentes recuperados por los algoritmos de búsqueda de los repositorios son diferentes, dependiendo del criterio utilizado (popularidad, vitalidad, clasificación, etc.). En el caso particular de AA y AB, se debe entender que los componentes seleccionados tienen las características más cercanas relacionadas con los requerimientos deseados de los componentes (Componente 3, 4, 6 o 10). Observando los resultados de los algoritmos de búsqueda de los repositorios, vemos que los componentes 21 y 11 tienen alto nivel de vitalidad, pero no tienen buena relación con los requerimientos deseados. Por otro lado, basado en la clasificación, el componente 17 sería la selección, y el 22 sería el mejor basado en la categoría; sin embargo, estos componentes no se relacionan tampoco con los requerimientos deseados. El AA genera resultados aceptables, sin embargo, el AB no solo genera buenos resultados, sino que logra buscar componentes en varios repositorios para obtener una mejor selección.

Experimento B:*Selección de 3 componentes:*

En esta prueba suponemos que se tiene un sistema compuesto por 3 componentes (los tres cuyos requisitos son especificados en la tabla 17). A continuación se muestran los resultados solo para los primeros dos componentes identificados en la tabla 17, la tabla 19 muestra los resultados obtenidos con el algoritmo AB, y la tabla 20 con el algoritmo AA.

Tabla 19: Resultados para algoritmo AB con 10 agentes (X= Valor no disponible).

Componentes encontrados		**FreshMeat**			**SourceForge**		**Algoritmo B Número de Agentes: 10**
		Popularidad	Vitalidad	Clasificación	Número de Descargas	Categoría	Probabilidad
Agente 1 C1	63	X	X	X	189	522	0.5
	34	X	X	X	0	33.6	0.08
	23	X	X	X	0	13918	0.05
	22	X	X	X	0	1.23	0.04
	69	X	X	X	170	22776	0.04
Agente 1 C2	Este agente no consiguió ese componente en ningún repositorio						
Agente 2 C1	63	176	11.4	0	X	X	0.21
	24	96	2.15	0	X	X	0.14
	28	226	16.5	0	X	X	0.08
	32	52	1.19	0	X	X	0.07
	18	437	117	0	X	X	0.05
Agente 2 C2	76	176	11.4	0	X	X	0.4
	18	437	117	0	X	X	0.1
	67	64	1	0	X	X	0.09
	59	232	822	8.41	X	X	0.09
	30	75	3	0	X	X	0.07
	24	X	X	X	189	522	0.5
	34	X	X	X	0	33.6	0.08

Agente 3 C1	23	X	X	X	0	13918	0.05
	22	X	X	X	0	1.23	0.04
	69	X	X	X	170	22776	0.04
Agente 3 C2	Este agente no consiguió ese componente en ningún repositorio						
Agente 4 C1	63	X	X	X	189	522	0.5
	34	X	X	X	0	33.6	0.08
	23	X	X	X	0	13918	0.05
	22	X	X	X	0	1.23	0.04
	69	X	X	X	170	22776	0.04
Agente 4 C2	67	64	1	0	X	X	0.22
	59	232	822	8.41	X	X	0.11
	30	75	3	0	X	X	0.09
	24	56	32	1.4	X	X	0.05
	12	20	2	0	X	X	0.04
Agente 5 C1	76	176	11.4	0	X	X	0.21
	24	96	2.15	0	X	X	0.14
	28	226	16.5	0	X	X	0.08
	32	52	1.19	0	X	X	0.07
	18	437	117	0	X	X	0.05
Agente 5 C2	76	176	11.4	0	X	X	0.4
	18	437	117	0	X	X	0.1
	67	64	1	0	X	X	0.09
	59	232	822	8.41	X	X	0.09

	30	75	3	0	X	X	0.07
Agente 6 C1	76	176	11.4	0	X	X	0.21
	24	96	2.15	0	X	X	0.14
	28	226	16.5	0	X	X	0.08
	32	52	1.19	0	X	X	0.07
	18	437	117	0	X	X	0.05
Agente 6 C2	76	176	11.4	0	X	X	0.4
	18	437	117	0	X	X	0.1
	67	64	1	0	X	X	0.09
	59	232	822	8.41	X	X	0.09
	30	75	3	0	X	X	0.07
Agente 7 C1	76	176	11.4	0	X	X	0.21
	24	96	2.15	0	X	X	0.14
	28	226	16.5	0	X	X	0.08
	32	52	1.19	0	X	X	0.07
	18	437	117	0	X	X	0.05
Agente 7 C2	Este agente no consiguió ese componente en ningún repositorio						
Agente 8 C1	76	176	11.4	0	X	X	0.21
	24	96	2.15	0	X	X	0.14
	28	226	16.5	0	X	X	0.08
	32	52	1.19	0	X	X	0.07
	18	437	117	0	X	X	0.05

Agente 8 C2	Este agente no consiguió ese componente en ningún repositorio						
Agente 9 C1	76	176	11.4	0	X	X	0.21
	24	96	2.15	0	X	X	0.14
	28	226	16.5	0	X	X	0.08
	32	52	1.19	0	X	X	0.07
	18	437	117	0	X	X	0.05
Agente 9 C2	Este agente no consiguió ese componente en ningún repositorio						
Agente 10 C1	76	176	11.4	0	X	X	0.21
	24	96	2.15	0	X	X	0.14
	28	226	16.5	0	X	X	0.08
	32	52	1.19	0	X	X	0.07
	18	437	117	0	X	X	0.05
Agente 10 C2	76	176	11.4	0	X	X	0.4
	18	437	117	0	X	X	0.1
	67	64	1	0	X	X	0.09
	59	232	822	8.41	X	X	0.09
	30	75	3	0	X	X	0.07

Tabla 20: Resultados para Algoritmo AA

Componentes Encontrados		**FreshMeat**			**SourceForge**		**Algoritmo A**
		Popularidad	Vitalidad	Clasificación	Número de Descargas	Categoría	Probabilidad
C1	24	X	X	X	189	522	0.5
	34	X	X	X	0	33.6	0.08
	23	X	X	X	0	13918	0.05
	22	X	X	X	0	1.23	0.04
	69	X	X	X	170	22776	0.04
C2	67	64	1	0	X	X	0.22
	59	232	822	8.41	X	X	0.11
	30	75	3	0	X	X	0.09
	24	56	32	1.4	X	X	0.05
	12	20	2	0	X	X	0.04

Cuando los componentes son encontrados en FreshMeat, los criterios usados para su valoración son popularidad, vitalidad y clasificación; y en el caso de SourceForge son número de descargas y categoría. Si el mismo componente existiese en FreshMeat y SourceForge, entonces tendrían sus valores para todos los criterios (Popularidad, Vitalidad, Clasificación, Descargas, Categoría).) Así, en las tablas 19 y 20, cuando un componente no se encuentra en ningún repositorio, no tendremos los valores de los criterios respectivos disponibles (se identifican como componente no encontrado). Por otro lado, la Probabilidad es el criterio usado/calculado por nuestros algoritmos para buscar/escoger nuestros componentes. Ella indica que tan bueno es cada componente de acuerdo a los requerimientos funcionales y no funcionales deseados, y al rendimiento de dicho componente en el dominio específico donde se quiere usar el mismo, llamado feromona (ver la sección 3 para más detalles al respecto).

El algoritmo AB da resultados más ricos que el AA, porque propone más opciones de componentes. Esto es debido a que existen mayor número de agentes buscando los mismos componentes, lo cual permite que existan mayor número de soluciones al problema inicial que es encontrar componentes con ciertas características particulares (por ejemplo, el agente 3 de AB propone los mismos C1 que el algoritmo AA, y el agente 4 de AB los mismos C2 que AA). Es decir, da una mayor diversidad de resultados. Ahora bien, vemos en este ejemplo que tanto el algoritmo AB como el AA pueden buscar en tantos repositorios como sean requeridos. También vemos que en algunos casos los agentes del AB no encuentran ninguna sugerencia para ciertos componentes, por ejemplo en el caso del agente 1 para el componente C2, entre otros. Esto es debido a que el repositorio en el que buscó dicho agente no contiene ningún componente con las características deseadas. También, en algunos casos más de un agente genera los mismos resultados para algunos componentes. El agente 10, por ejemplo, selecciona los mismos componentes que el agente 2 cuando deben buscar los componentes 1 y 2. En general, para ambos algoritmos mientras más repositorios usen existe más posibilidad que consigan los componentes buscados.

Por otro lado, hicimos experimentos sobre el algoritmo AB para determinar cuál es el número idóneo de agentes a crear para realizar las búsquedas (ver tabla 21). Mientras más agentes usan el AB, hasta un número dado, más soluciones diversas son provistas, por lo que hay más posibilidad de encontrar el mejor componente según los requerimientos deseados. De este experimento podemos deducir que en AB cuando son usados menos de 30 agentes para seleccionar quince componentes, algunas soluciones no son presentadas (por lo que existe la posibilidad de no encontrar el mejor componente); y cuando se usan más de 30 agentes para seleccionar los mismos quince componentes, muchas soluciones son repetidas, pero entre ellas aparece la mejor. Por otras pruebas realizadas para el AB semejantes a la anterior, se ha notado que un número de agentes efectivo seria el doble del número de componentes a buscar, esto significa que si se requiere la búsqueda de 20 componentes el número de agentes sugerido para generar buenos resultados seria 40. En las pruebas de la tabla 21 se corrió el algoritmo AB 30 veces para cada caso diferente estudiado. Las especificaciones de los componentes para cada caso fueron diferentes, con eso se hizo el análisis estadístico que permita identificar cual sería un buen número de agentes a crear para ejecutar el Algoritmo AB.

Tabla 21: Número de Agentes Vs. Número de Componentes a Buscar

Número de Agentes Utilizados	Número De Componentes	Resultado
6	3	Todos los Componentes Encontrados
10	5	Todos los Componentes Encontrados
30	15	Todos los Componentes Encontrados
3	3	No Todos los Componentes Encontrados
5	5	No todos los Componentes Encontrados
7	15	No todos los Componentes Encontrados

Los experimentos relacionados con los algoritmos de búsqueda demuestran que la suposición inicial de que estos generarían resultados favorables, e incluso mejores que otros algoritmos equivalentes, es correcta. Esto induce a pensar que esta propuesta puede llegar a ser una excelente solución a muchos de los problemas de selección y búsqueda de componentes de software. Así mismo, de la tabla 21 se puede inferir que un buen número de agentes a utilizar con el Algoritmo AB sería el doble del número de componentes a buscar.

5.3 Implementación del Middleware Reflexivo

Como requisito fundamental para usar el middleware, es requerido que la aplicación base esté estructurada por componentes de software que puedan reemplazarse o modificarse de manera independiente. Esto último es tomado en cuenta en el modelo Fractal (ver capítulo 2), que modela el diseño de arquitecturas basadas en componentes de software, modulares, extensibles e independientes al lenguaje de programación utilizado. Nuestro middleware reflexivo es implementado bajo esa filosofía,

tal que el nivel meta planteado por dicho modelo es nuestro middleware reflexivo (ver figura 23). La figura24 muestra como un grupo de componentes (AO) de una aplicación, es gestionado por nuestro middleware. Como se dijo antes, el middleware está implantado en el nivel Meta, donde la supervisión, monitoreo y manejo del ambiente comunicacional son tareas de los agentes principales del middleware: Reflector, Monitor e Info, respectivamente. Básicamente, lo que ocurre es que el middleware supervisa la aplicación base, para ello utiliza archivos XML que almacenan información sobre el funcionamiento de la misma (los agentes monitor e info se encargan de capturar los datos y gestionar dichos archivos, respectivamente), el middleware utiliza esta información para tomar decisiones sobre si es necesario hacer cambios a la aplicación base para mejorar su comportamiento (lo hace el agente reflector), de acuerdo a los criterios objetivos definidos en las configuración inicial del middleware.

Particularmente, en la figura 23 se ve reflejado con una flecha amarilla, la comunicación entre el nivel meta y el nivel base, la cual, como se dijo anteriormente, se hace a través de archivos XML que almacenan estados e información importante sobre la aplicación, para que el middleware (Meta) pueda ejecutar los cambios específicos a nivel de los componentes que conforman la aplicación base. De esta forma se mantiene la autonomía del nivel base, haciéndose su supervisión en el nivel meta.

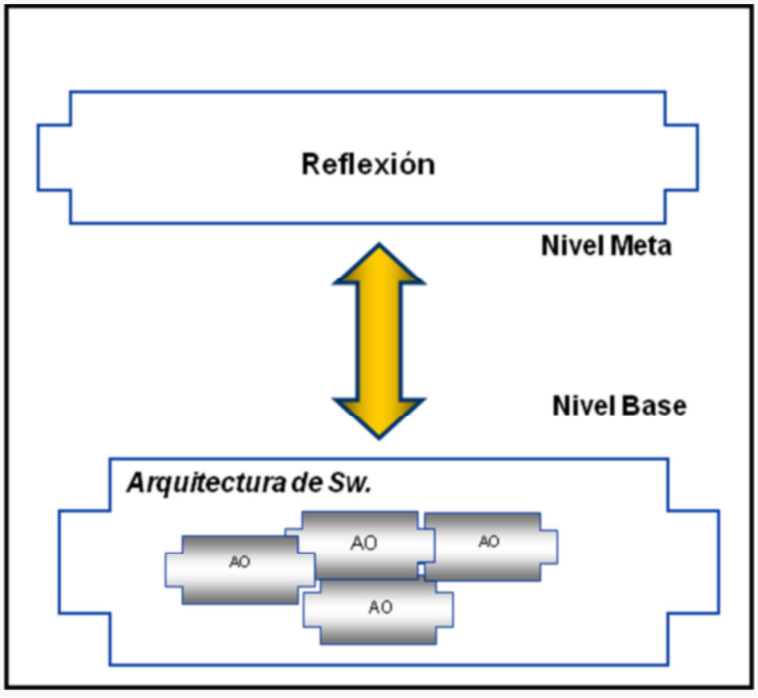

Figura 23: Modelo de Implementación para el Middleware Reflexivo sobre un sistema de componentes

En cuanto a la figura 24, la flecha que va del nivel base al meta refleja la comunicación entre ambos niveles. La flecha que va del nivel meta a ella misma refleja la comunicación entre los agentes que conforman el middleware. Por otro lado, la flecha que sale de C1 significa un cambio que genera el componente 1 en la aplicación base, y que a su vez genera un evento que el nivel meta requiere monitorear, y la flecha que entra a C1 significa una llamada que el nivel base ejecuta al componente 1, que lo activa. Dicha activación implica la ejecución de un proceso interno específico de C1.

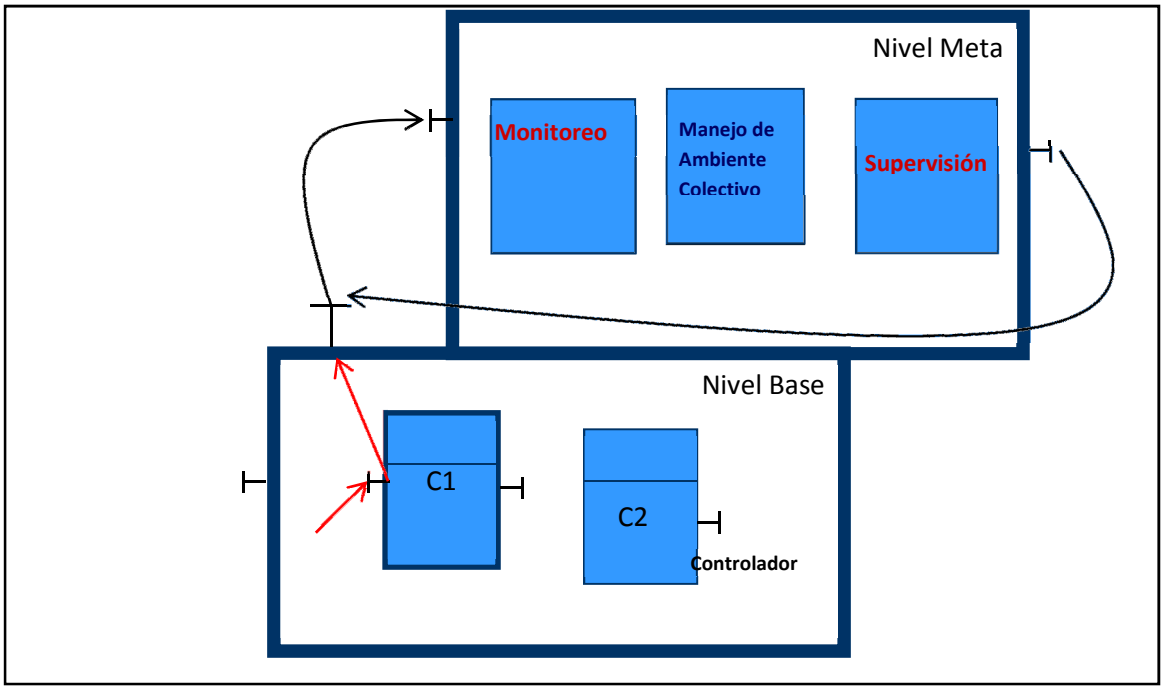

Figura 24: Middleware reflexivo utilizando Fractal como base de modelado

5.4 *Middleware Reflexivo para supervisar la seguridad en una aplicación basada en componentes.*

Como se dijo anteriormente cada componente de software está caracterizado, no solo por aspectos funcionales, sino también por aspectos no funcionales. Entre los aspectos no funcionales que se podrían definir para un componente están los niveles de confianza, lo cual se refiere a cuanto el programador puede confiar en un componente especifico, y que tan peligroso puede ser su incorporación a un

sistema. Específicamente, cuando se evalúan aspectos relacionados a seguridad en un componente los criterios a considerar son los de confidencialidad, integridad, disponibilidad, entre otros [72].

El middleware reflexivo propuesto en esta tesis se ha utilizado para manejar las características relacionadas con seguridad que cada componente de software posee, y como estas afectan a otros componentes cuando se integran en una aplicación. La idea principal del middleware es monitorear (también se conoce como introspección [11, 31, 69]) las aplicaciones, para reemplazar o ajustar (también conocido como intercesión [11, 31, 69]) los componentes de software, con el objetivo específico de hacer el sistema tan seguro como sea posible. El middleware es suficientemente flexible para ser configurado de acuerdo a las necesidades de la aplicación que será monitoreada, y hacer los cambios necesarios cuando la seguridad de la misma decrezca.

Los niveles de seguridad requeridos por la aplicación son definidos antes de su ejecución, tal que el middleware pueda tomar decisiones en tiempo de ejecución. Estas decisiones son ajustar o reemplazar uno o más componentes de la aplicación, y/o generar una alarma. Cuando el middleware ajusta un componente, el mismo puede, por ejemplo, cambiar su mecanismo de encriptamiento; si por el contrario, se requiere el reemplazo del componente, el middleware busca uno nuevo que posea las mismas funcionalidades y se realiza el cambio. Finalmente, el middleware puede generar una alarma a un grupo de usuarios o administradores del sistema, para comunicarles el cambio o ajuste que ha sido hecho, o notificar un nivel de seguridad no deseado en la aplicación.

El middleware toma la decisión de ajustar, reemplazar y/o generar una alarma, basado en que tanto haya decrecido el nivel de seguridad en comparación con los niveles especificados en la configuración inicial. De esta manera, nuestro middleware no solo audita automáticamente la seguridad de un sistema que está compuesto por varios componentes de software, sino que también hace cambios a los componentes cuya seguridad se vea comprometida. Esto es particularmente importante en sistemas grandes que están compuestos por un número significativo de componentes.

5.4.1. Planteamiento del problema de Seguridad.

Cuando una aplicación es desarrollada utilizando diferentes componentes de software, existen aspectos que deben ser considerados para mantener niveles de seguridad aceptables, especialmente, cuando estos componentes manipulan datos de archivos de texto, bases de datos, o se comunican. Así, el middleware propuesto debe evaluar los componentes de software cuando los mismos manipulan datos, y cuando los mismos se comunican entre sí. Existen tres aspectos que el middleware debe manejar con respecto al tema de seguridad en un componente, estos son [72]:

Confidencialidad: Es la propiedad de un componente para que la información que manipula solo pueda ser accedida de manera segura por los usuarios o componentes que él autorice (garantiza que solo quien esté autorizado puede ver dicha información, usando para ello mecanismos de cifrado, etc.).

Integridad: Esta propiedad garantiza que los datos sean modificados coherentemente por los componentes y usuarios autorizados por el componente que controla el acceso a dichos datos. Para ello se requieren de mecanismos que garanticen que las modificaciones sean hechas correctamente (de coherencia, de acceso atómico a los datos, etc.).

Autenticación: Este es el proceso que permite la verificación de la identidad de un usuario o componente que requiere interactuar con el componente de software.

El middleware permite mantener el sistema tan seguro como sea posible, incorporando estas características de manera dinámica. El middleware asegura:

- La integridad de la información que está siendo modificada y compartida por diferentes componentes.
- La autenticación de los usuarios o componentes que están autorizados a interactuar con un componente de software dado.
- La confidencialidad de la información para que no sea vista por intrusos.

Cada componente de software que integra la aplicación estará caracterizado por esos tres aspectos de seguridad nombrados antes, los cuales fueron seleccionados por ser algunos de los aspectos más comúnmente monitoreados en esta área [72]; definiéndose los niveles de seguridad según lo indicado en la tabla 22. Cada aspecto tendrá un número entre 0 y 2, donde 2 denota el nivel de mayor seguridad

del componente para ese aspecto. Una combinación de cada aspecto de seguridad puede ser definida por cada componente del sistema, con los niveles de seguridad requeridos para cada uno de ellos. Los requerimientos de seguridad de cada componente conformarán lo que se llamará la configuración inicial del sistema.

El sistema tendrá el mismo nivel de seguridad que el componente más inseguro que lo integra. Dicho sistema deberá mantener un nivel de seguridad mínimo establecido en la configuración inicial del middleware. Si este valor disminuye por debajo del nivel especificado, el middleware reemplazará o ajustará a uno o más componentes, para volver a estabilizar el sistema. El middleware monitorea el comportamiento de los componentes, y evalúa si los mismos cumplen con la configuración inicial del sistema.

Tabla 22: Descripción de niveles de seguridad relacionados a cada requerimiento

Nivel de Seguridad	Confidencialidad	Integridad	Autenticación
0	No usa encriptamiento.	No posee mencanismos para asegurar la integridad de la información.	Componente puede interactuar con cualquier usuario/componente.
1	Algún encriptamiento es necesario – No existe un mecanismo de encriptamiento predefinido	Posee procesos básicos para asegurar la integridad de la información, sin validar si funcionan correctamente.	Acceso al componente está relacionado a cada usuario.
2	Un mecanismo fuerte de encriptamiento.	Se requieren de mecanismos para asegurar la integridad de la data. Además, el sistema provee mecanismos de validación y auditoría.	Acceso al componente está relacionado con cada usuario, y un log de las actividades de cada usuario es mantenido.

5.4.2. Implementación

El primer caso de estudio para probar la implementación del middleware reflexivo ha sido enfocado a la seguridad relacionada con los componentes que conforman una aplicación base. Se desea probar que mediante la utilización del middleware reflexivo propuesto se puede mantener la seguridad de la aplicación base de manera automática.

Debido a que este caso de estudio se enfoca en seguridad, el middleware fue configurado para verificar la confidencialidad, integridad y autenticación de una aplicación específica. Este middleware reflexiona sobre la arquitectura del sistema, permitiendo una selección dinámica de nuevos componentes de software basada en requerimientos de seguridad. Las preguntas principales son: ¿Cómo traer un nuevo componente a un sistema catalogado como seguro, sin que esto haga que se pierda dicho nivel de seguridad?, ¿Cómo determinar la confiabilidad del nuevo componente a incorporar?, ¿Cómo seleccionar el componente más seguro?, ¿Cómo cambiar un componente para mejorar la seguridad del sistema?.

Los agentes del middleware, y tareas que los mismos realizan, son las siguientes:

Monitor: Este es un agente que revisa si cada componente del sistema posee los requerimientos de seguridad necesarios. Si se determina que este no es el caso, un mensaje es enviado al reflector, a través de los archivos XML. Esto ocurre cuando se detecta una falla de seguridad (alguno de sus mecanismos, ya sea el de encriptamiento, integridad o autenticación, ha sido violado).

Reflector: Este agente toma la decisión acerca de que acción realizar. Actualmente, las posibles decisiones son: Cambiar uno o más componentes, ajustar un componente específico (el mecanismo de encriptamiento, o el mecanismo de integridad, o el mecanismo de autenticación usado, según sea la falla de seguridad detectada), o solo generar una alarma indicando que la seguridad de la aplicación está por debajo de la aceptada. La “experiencia” sobre el problema encontrado, y la manera en la que fue resuelta, es almacenada en un archivo XML. Específicamente, se mantienen registros sobre la fecha y hora en la que se ha encontrado el problema, el tipo de problema, y la solución generada.

Manejador de Información: Este agente almacena toda la información generada luego de cada cambio, y durante la ejecución de la aplicación.

En este caso de estudio, el objetivo es encontrar los mecanismos idóneos (según la falla de seguridad)

que logren mantener a la aplicación segura. Para ello, se requirieron desarrollar nuevos métodos para los agentes especializados de nuestro middleware, vinculados a los temas de seguridad a monitorear en las aplicaciones, algunos de los cuales son:

```
//Se cargan todas las variables configurables de la libreria
// Variables de seguridad y de manejo de componentes
configuration.set_Security();
//Analiza la seguridad relacionada a un componente
reflect.analizing_security(current_used_component);
 //Cambia mecanismo de encriptamiento
changeEncryption("Component_Name");
```

El monitor revisa los posibles problemas de seguridad de manera proactiva, según lo especificado en los requerimientos iníciales. Este módulo ha sido configurado para correr cada cinco minutos. El Manejador de Información maneja los archivos XML que tienen la información relacionada a los componentes de software (Fig. 25). Por ejemplo "name" es el nombre del componente, "path" es la localidad del componente, "user" es el nombre del usuario o componente que tiene permisos para manipular el componente (usando esta variable se gestiona la autenticación, confiabilidad e integridad de un componente dado), "rights" especifica los permisos para cada usuario, y "levels" los niveles de seguridad del componente. En general, cada usuario y componente tendrá permisos que se especifican en la etiqueta "rights"; esta etiqueta especifica tres números, cada uno puede ser 0 o 1 (0 significa que no tiene permiso, 1 significa que si posee el permiso). El primer número es el permiso de lectura, el segundo número identifica el permiso de escritura, y el tercer número se relaciona a la capacidad de ejecución. Por ejemplo, en la figura 25 el XML define el usuario "Blanca Abraham", quien no tiene ninguna restricción de uso. Por otro lado, la etiqueta "level" define los niveles de seguridad del componente, según lo descrito en la tabla 6. Este XML (fig. 25) muestra la configuración mínima que el middleware necesita para funcionar, sobre los aspectos de seguridad de cada componente. El Reflector le provee información al Manejador de Información luego de tomar las decisiones finales, y después de evaluar cuáles son los mecanismos de seguridad más eficientes para cada componente en particular.

Adicionalmente, el Manejador de Información almacena todas las acciones que el usuario realiza con los

componentes, y las que el middleware realiza dentro del sistema, como ajustes o cambios de componentes, en un XML, al cual se le denomina histórico (fig.26). Algunas de las informaciones almacenadas tienen que ver con los usuarios que tratan de acceder al componente, el tipo de acceso que es realizado, las alarmas que se generan, y las acciones que el usuario o el middleware realizan. En el ejemplo de la figura 26 se puede ver que el día 08/03/03, a las 16:15:06 horas, una alarma fue generada y una acción fue hecha, en este caso, el reemplazo de un componente (RP significa reemplazo), otras posibles acciones se mencionan más adelante.

Las diferentes acciones que el middleware puede generar son:

Alarma/Mensaje: Este es un elemento configurable del middleware, el cual puede ser un email que el middleware envía a un grupo de usuarios, un mensaje en pantalla. La etiqueta "alarm" de la figura 26 será 1 si se ha enviado una alarma, o 0 si no se ha enviado.

Ajuste (T): En este caso, el middleware cambia el mecanismo de seguridad (de encriptamiento, de integridad, de autenticación) de un componente si existe una falla de seguridad.

Reemplazo (RP):Cuando un ajuste no puede ser hecho, el middleware trata de reemplazar el componente, y si esto no es posible, el mismo hace la sugerencia de reemplazo al administrador del sistema responsable de la aplicación que el middleware está monitoreando. Si el middleware trata de reemplazar el componente, los algoritmos AA o AB son llamados para buscar y seleccionar el mejor componente que reemplazará a aquel con problemas.

En general, el sistema está bajo ataque cuando la seguridad de un componente se rompe, por ejemplo, cuando usuarios o componentes no autorizados logran interactuar con un componente dado. En estos casos, el Monitor manda un mensaje a través de un archivo XML al Reflector (quien actualiza dicho archivo es el agente Info), el cual lo activará, para tomar la decisión de si un ajuste o reemplazo es necesario. En general, el Manejador de Información es el que gestiona los archivos XML, y mantiene actualizada la información con todo lo que acontece en el middleware, y con la información que le pasa el agente Monitor.

```
<MIDDLEWARE CONFIGURATION>03-03-08</MIDDLEWARE CONFIGURATION>
<Application>
<AppName>Reflective Architecture Test<AppName>
<SoftwareComponent>
      <Name>Geometry</Name>
<Path>c:/componentloc/geom</Path>
      <Security>
            <User>BLANCA ABRAHAM</User>
            <Rights>111</Rights>
            <Levels>
                  <Confidenciality>0</Confidenciality>
                  <Integrity>0</Integrity>
                  <Authentication>0</Authentication>
            </Levels>
      </Security>
      <Performance>
      </Performance>
</SoftwareComponent>
<SoftwareComponent>
      <Name>ReportMng</Name>
<Path>c:/componentloc/RM</Path>
      <Security>
            <User>app</User>
            <Rights>101</Rights>
            <Levels>
                  <Confidenciality>1</Confidenciality>
                  <Integrity>1</Integrity>
                  <Authentication>1</Authentication>
            </Levels>
      </Security>
```

Figura 25: XML de Configuración del Middleware

```
<MIDDLEWARE CONFIGURATION>03-03-08</MIDDLEWARE CONFIGURATION>

<Date>08-03-03:16:15:05</Date>
<User>BLANCA ABRAHAM</User>
<Alarm>1<\Alarm>
<Action>RP<\Action>

<Date>08-03-03:16:20:13</Date>
<User>BLANCA ABRAHAM</User>
<Alarm>0<\Alarm>
<Action>RP<\Action>

<Date>08-03-03:11:19:32</Date>
<User>TEST USER</User>
<Alarm>1<\Alarm>
```

Fig. 26: Archivo con información histórica status.xml

5.4.3. Experimentación

En este caso se tomó una aplicación que consiste en cuatro componentes. Se implementaron dos pruebas, la primera consiste en que todos los componentes se comunican entre ellos (PA), y en la segunda prueba los componentes no se comunican (PB).

Los requerimientos de seguridad iniciales definidos para probar la aplicación son definidos en la Tabla 23:

- Componente A: No tiene restricciones de seguridad, es un componente que no requiere seguridad.
- Componente B: Tiene un nivel de seguridad medio para todos los niveles de seguridad.
- Componente C: Tiene altos niveles de seguridad, por esta razón el middleware generara alarmas, ajustes y reemplazos si la seguridad se compromete.

- Componente D: Tiene variados requerimientos de seguridad. No requiere seguridad a nivel de autenticación, pero si requiere un nivel alto para el aspecto de confidencialidad, e intermedio para la integridad.

Tabla 23: Descripción de los niveles de seguridad para el caso de estudio

Componente	Nivel de Confidencialidad	Nivel de Integridad	Nivel de Autenticación
A	0	0	0
B	1	1	1
C	2	2	2
D	2	1	0

Para el caso de PA, debido a que todos los componentes se comunican los niveles de seguridad se auto-regulan al más alto. En este caso, el componente C es el que posee el nivel de seguridad más alto, y por tanto, el resto de los componentes deben mantener el mismo nivel para garantizar las restricciones de seguridad. Esto significa que el nivel de seguridad total de un sistema con componentes que se comunican entre sí será el nivel más alto de entre los componentes que lo integran.

5.4.4. Análisis de Resultados

En la tabla 24 se muestran los ataques que se simulan para cada componente de la aplicación; en este caso son tres ataques por componente por tipo de característica de seguridad.

Tabla 24: Número de ataques para cada componente de la aplicación base

Componente	Ataques de Confidencialidad	Ataques de Integridad	Ataques de Autenticación
A	3	3	3
B	3	3	3
C	3	3	3
D	3	3	3

5.4.5. Prueba para el caso PA:

La siguiente tabla muestra las alarmas que son generadas por el Middleware para PA, cuando la seguridad de cada componente está bajo un ataque, y a su vez los componentes se comunican. Las alarmas/mensajes generados por el middleware para cada componente son las siguientes (ver Tabla 25):

Tabla 25: Las alarmas/mensajes generados por el middleware para cada componente – caso PA.

Componente	Número de Alarmas para Confidencialidad	Número de Alarmas para Integridad	Número de Alarmas para Autenticación
A	3	3	3
B	3	3	3
C	3	3	3
D	3	3	3

Debido a que en PA todos los componentes se comunican, cada vez que alguno de ellos está bajo cualquier ataque una revisión debe ser hecha, aun si el componente no la requiere porque su nivel de seguridad es bajo.

Adicionalmente, cuando uno de los componentes está bajo ataque, todos los niveles de seguridad del resto de los componentes deben ser revisados. El número de alarmas significa el número de veces que

el middleware envía un mensaje (email, escritura en el log, reporte en pantalla, etc.) al administrador del sistema. La alarma es generada cada vez que el middleware encuentra algo que no es esperado, según los requerimientos iníciales de seguridad; es importante recordar que el middleware corre como un proceso que está monitoreando el sistema base cada cinco minutos (este tiempo es configurable).

En el caso del componente A el mismo genera alarmas porque aun cuando el nivel de seguridad que requiere es pequeño, el mismo tiene comunicación con otros componentes que poseen mayor nivel de seguridad, por tanto, al estar bajo ataque se considera que los componentes con los que se comunica también lo están. En este caso se permitió que el Monitor y el Reflector mostraran en pantalla cada acción que ejecutaron para mostrar el funcionamiento del middleware. Por ejemplo, la figura 27 muestra las alarmas cuando se encuentra un usuario leyendo (o que leyó) un archivo no autorizado (sin usar el componente que gestiona el encriptamiento/desencriptamiento del archivo). En este caso se intenta ajustar el componente cambiando el algoritmo de encriptamiento; a fin de que los archivos que el componente manipula no puedan ser "leídos" por componentes o usuarios no autorizados.

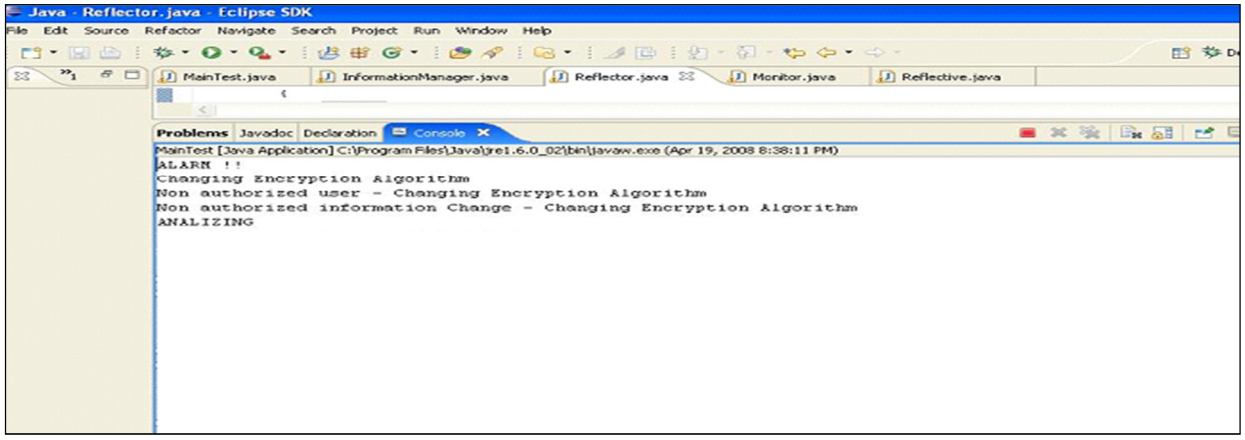

Figura 26: Middleware monitoreando y generando alarmas al encontrar inusual comportamiento (ver anexo 1 que contiene el manual de uso del middleware).

La tabla 26 muestra el número de ajustes que fueron requeridos cuando los componentes de seguridad estuvieron bajo ataque. En general, debido a que en PA todos los componentes se comunican, entonces existe un mayor número de ajustes. Por ejemplo, el componente A posee un nivel de confidencialidad, integridad y autenticación de 0, sin embargo, debido a que este componente se comunica con el resto, el comportamiento esperado seria que cuando el mismo está bajo ataque un ajuste debe ser hecho para

mantener el nivel de seguridad de los componentes asociados a este. Este mismo comportamiento fue ejecutado por el middleware, probando la efectividad del mismo para este tipo de configuración. La diferencia entre el número de ajustes se debe a que en ocasiones el middleware "decide" hacer ajustes y otras veces "decide" solo generar una alarma, o reemplazar al componente (si existen otras versiones del componente disponibles, o un componente nuevo que haga lo mismo y lo pueda sustituir, con mejores niveles de seguridad).

Tabla 26: Ajustes realizadas a cada componente – Todos los componentes se comunican

Componente	Número de Ajustes debido a Confidencialidad	Número de Ajustes debido a Integridad	Número de Ajustes debido a Autenticación
A	1	2	3
B	2	1	2
C	3	3	2
D	2	2	3

El número de reemplazos necesarios para PA son mostrados en la Tabla 27, los reemplazos corresponden a los ataques simulados para cada componente, según el nivel de seguridad que el mismo tenga configurado; así se demuestra la efectividad del middleware, el cual es capaz de llamar al algoritmo de búsqueda cada vez que es requerido un reemplazo, para proponer los componentes más apropiados a utilizar. Dichos reemplazos son realizados por el administrador del sistema cuando existe una versión nueva del componente que incremente su nivel de seguridad, o un componente con las mismas funcionales y el nivel de seguridad requerido.

Tabla 27: Reemplazos realizados para cada componente

Componente	Número de Reemplazos por Confidencialidad	Número de Reemplazos por Integridad	Número de Reemplazos por Autenticación
A	2	1	0
B	1	2	1
C	0	0	1
D	1	1	0

Finalmente, se puede decir que para el caso de seguridad el objetivo inicial de mantener los valores de seguridad de una aplicación base de manera automática, según la configuración inicial requerida, ha sido logrado a través de la implementación del middleware reflexivo. De esta manera demostramos que procesos de mantenimiento de sistemas, tan sensibles a temas como los de seguridad, pueden ser automatizados tal como se mostró con este caso de uso.

5.4.6. Prueba para el caso PB

En PB no todos los componentes se comunican. Los resultados para esta prueba son mostrados en las siguientes tablas. La tabla 28 muestra las alarmas cuando la seguridad de los componentes está comprometida y no existe comunicación entre ellos. Esto significa que si un componente está bajo ataque no pone en riesgo a otro componente, pues no existe comunicación entre ellos.

Tabla 28: Alarmas generadas por el middleware – No existe comunicación entre componentes

Componente	Número de Alarmas por Confidencialidad	Número de Alarmas por Integridad	Número de Alarmas por Autenticación
A	0 – No Alarma	0 – No Alarma	0 – No Alarma

B	2	3	2
C	3	3	2
D	3	1	0 – No Alarma

A diferencia de PA, en PB el componente A, por ejemplo, nunca genera una alarma cuando es atacado, pues su nivel de seguridad es 0 (ver tabla 23), porque no se comunica con otro componente. Para el componente B, el cual tiene un nivel de seguridad de 1 para todos los aspectos, las alarmas son enviadas cada vez que se genera un cambio. Para el componente C, el cual tiene el nivel más alto de seguridad, se envían alarmas cuando su confidencialidad, autenticación o integridad han sido violadas. Finalmente, el componente D posee diferentes niveles de seguridad, las alarmas son enviadas solo cuando problemas relativos a confidencialidad o integridad son detectados. Como podemos ver, estos valores de la tabla 28 dependen de la configuración inicial.

La tabla 29 muestra los ajustes al simular ataques de confidencialidad, integridad y autenticación para cada componente. En este caso, el componente A no tiene ajustes pues este no requiere protección y no posee comunicación con otros componentes en esta prueba. Para el componente B solo se hizo un ajuste por confidencialidad, esto se debe a que en los otros dos ataques se generaron simplemente alarmas. Un comportamiento similar se encuentra en el componente C, en el cual se generan dos ajustes por confidencialidad y tres alarmas por confidencialidad, esto significa que aun cuando en una ocasión se generó una alarma no existió ningún ajuste que pudiera ser realizado (ver Tablas 28 y 29).

Tabla 29: Ajustes que fueron hechos en cada componente– No existe comunicación entre los componentes

Componente	**Número de ajustes por Confidencialidad**	**Número de ajustes por Integridad**	**Número de ajustes por Autenticación**
A	0	0	0
B	1	3	2
C	2	2	1

D	3	1	0

Por otro lado, los componentes de software pueden tener nuevas versiones disponibles que podrían ser usadas, especialmente cuando un componente no está funcionando apropiadamente, o pueden existir componentes que cumplan la misma función de manera más segura. Los cambios que el middleware hizo en cada componente son mostrados en la tabla 30. En este caso, solo los componentes B, C y D sufrieron reemplazo para mantener sus niveles de seguridad en lo requerido.

Tabla 30: Reemplazos sugeridos por cada componente

Componente	Número de reemplazos por Confidencialidad	Número de reemplazos por Integridad	Número de reemplazos por Autenticación
A	0	0	0
B	1	1	0
C	1	1	1
D	1	0	0

La posibilidad del reemplazo del componente por una nueva versión, o un componente que haga lo mismo, es evaluada como opción, si un ajuste no permite resolver el problema de seguridad.Para esto último, los algoritmos de selección detallados en el capítulo 3 son utilizados.

Cuando una nueva versión del mismo componente no se encuentra, un componente diferente, con las mismas características funcionales y no funcionales, es seleccionado como reemplazo. Los reemplazos de componentes requieren la intervención del administrador del sistema, debido a que esta acción requiere la configuración de la búsqueda y la integración del nuevo componente con el resto de la aplicación. La figura 28 muestra una invocación del middleware al sistema de búsqueda propuesto en el capítulo 3. La ventana pequeña muestra los requerimientos introducidos para la búsqueda, y la grande el proceso de búsqueda (ver anexo 1 que contiene el manual de usuario de los algoritmos de búsqueda).

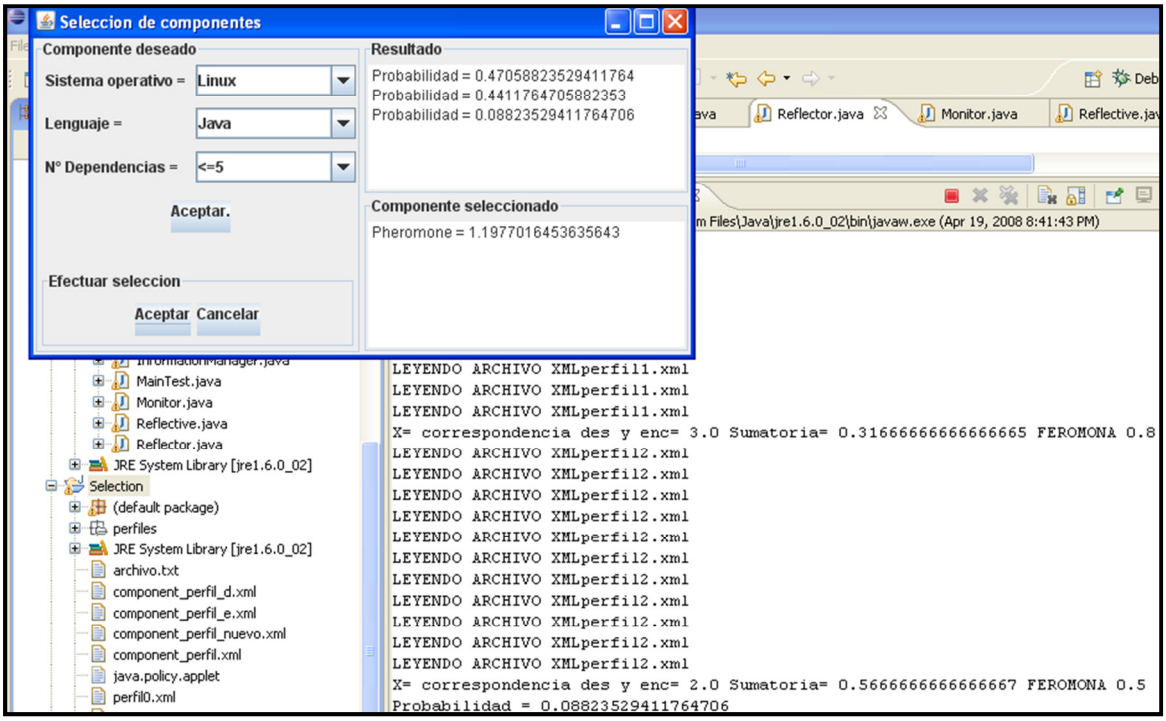

Figura 27: Middleware buscando nuevo componente

En resumen, el componente B ha sido reemplazado dos veces (ver tabla 30), una debido a problemas de integridad de la información y otra por problemas de confidencialidad. Este mismo componente ha sido ajustado seis veces (tabla 29); una vez por problemas de confidencialidad, tres veces por problemas de integridad y dos veces por problemas de autenticación. Este comportamiento es similar para los otros componentes.

En el caso de seguridad, cuando los componentes de software se comunican entre ellos, el middleware también cumple su función de mantener la seguridad de la aplicación base, tal como se esperaba.

5.5 Middleware Reflexivo para supervisar el rendimiento en una aplicación basada en componentes.

En este caso de estudio la implementación del middleware reflexivo ha sido enfocada a rendimiento de

los componentes que conforman una aplicación base. Se desea probar que mediante la utilización del middleware reflexivo propuesto se puede mantener un rendimiento aceptable de la aplicación base de manera automática.

Para este caso de estudio, el middleware supervisa los posibles problemas de rendimiento. Para esto, se configura que el monitoreo se ejecute cada 30 minutos. Con la finalidad de responder a cada evento que el Monitor encuentra, tres acciones pueden ser ejecutadas por el Reflector, estas son:

Alarma: Es un elemento configurable en el middleware, puede ser un email o un mensaje enviado a un grupo de usuarios o administradores del sistema.

Ajuste:Es la reflexión que el middleware realiza sobre la aplicación base. En este caso, el middleware (Reflector) ejecuta una serie de acciones para mejorar el rendimiento de la aplicación. Dichas acciones son:

- Liberar memoria del sistema.
- Liberar espacio de disco duro.
- Liberar utilización de CPU.

Reemplazo: Cuando un ajuste no es suficiente para mejorar el rendimiento de la aplicación, el middleware sugiere al administrador del sistema el reemplazo del componente. El reemplazo del componente puede ser hecho por una versión más nueva, si esto es posible, o por otro componente con las mismas funcionalidades. En este caso se usa el algoritmo de búsqueda de componente (AB) presentado en el capítulo 3.

Por otro lado, se requiere de un agente especializado que realiza, entre otras cosas, las siguientes tareas:

```
// Calcula número de procesadores existentes
processorsNum = special.availableProc();
// Calcula Memoria RAM libre
freeMem = special.freeMem();
// Calcula Memoria RAM que existe en el ambiente de ejecución.
totalMem =  special..totalMem();
// Calcula Memoria que utiliza el sistema base
```

```
usedMem = special.used_memory();
// Calcula Memoria que usa cada componente
compusedMem = used_memory("Componente");
//Libera Memoria RAM
release_mem();
//Libera CPU
stop_Proc();
//Libera disco duro
release_hd();
```

5.4.7. Experimentos

La aplicación a estudiar está compuesta por tres componentes; cada uno de ellos tiene sus propios requerimientos de rendimiento; los mismos están relacionados con memoria, tiempo de procesamiento, y disco duro. El Monitor verificara lo siguiente:

- Uno o más componentes están consumiendo más memoria de lo permitido.
- El requerimiento de disco duro no está disponible.
- El tiempo de procesamiento requerido está por encima de la capacidad del CPU.

Se ha configurado el middleware para monitorear y tomar decisiones en base a tres niveles de rendimiento y prioridad de asignación de recursos:

Nivel 0: para componentes con poca prioridad; en este caso, el middleware no le asignara recursos a menos que estos estén disponibles.

Nivel 1: Se asignan recursos a componentes con prioridad media.

Nivel 2: Se asignan recursos a los componentes que tienen la mayor prioridad.

Si un componente posee nivel de prioridad 2, el middleware tomará las decisiones necesarias para conseguir los recursos que este componente requiere para funcionar. Estas acciones son: Liberar memoria RAM, liberar uso de CPU, liberar espacio de disco duro. La idea principal del middleware es asegurar que cada componente de la aplicación base cuente con los recursos que requiere para su

ejecución, lo cual es sumamente útil cuando se tratan sistemas que deben mantenerse bajo ejecución ininterrumpida. En la tabla 31 se muestra la configuración inicial del sistema:

Tabla 31Configuración inicial a mantener para la Aplicación Base:

Componente	Nivel para garantizar rendimiento óptimo, RAM	Nivel para garantizar rendimiento óptimo, CPU	Nivel para garantizar rendimiento óptimo, Disco Duro
A	0	0	1
B	0	0	0
C	2	2	2

En la tabla 32 se muestra el número de ajustes realizados. Por ejemplo, para el componente 'A', el cual posee una prioridad 0 para RAM y CPU, no se hacen ajustes para liberar recursos pues su prioridad es muy baja; así mismo, para el componente B; sin embargo, para el componente C, cuya prioridad es bastante alta, el Middleware busca la manera de liberar todos los recursos posibles, a fin de permitir que este componente funcione permanentemente con los recursos requeridos.

Tabla 32: Número de Ajustes

Componente	Cambios relacionados a Memoria - RAM	Cambios relacionados a CPU	Cambios relacionados a Disco Duro
A	0	0	1
B	0	0	0
C	4	2	3

El middleware envía una alarma cada vez que un componente requiere recursos (ver tabla 33). Por ejemplo, para el componente 'A', cuyo nivel de prioridad para RAM y CPU es 0, se genera un mayor número de alarmas que ajustes. Para el componente 'C', que tiene el nivel más alto de prioridad, los recursos son asignados cada vez que se requieren, aun si esto amerita tomar recursos de otro componente. Por esta razón, para el componente 'C' el número de alarmas generadas es igual al número de ajustes realizados.

Tabla 33: Alarmas generadas por el middleware cuando son requeridos recursos

Componente	Requerimientos de Asignación Memoria RAM	Requerimientos de Asignación CPU	Requerimientos de Asignación Disco Duro
A	17	15	3
B	17	14	2
C	4	2	3

El reemplazo de los componentes del sistema base se muestra en la tabla 18. Los reemplazos se ejecutan cuando, luego de asignar los recursos disponibles, el componente todavía no logra ejecutar sus tareas. Para ubicar y seleccionar los mejores componentes para realizar el reemplazo, se utilizan los algoritmos de búsqueda explicados en detalle en el capítulo 3. Solo unos pocos reemplazos fueron ejecutados, debido a que los problemas de rendimiento en general pudieron resolverse a través de los ajustes (consistió en liberar recursos para que los componentes pudieran ejecutar las acciones esperadas). En este caso, el reemplazo del componente 'C' fue necesario, pues aun luego de asignarle más CPU y memoria, todavía el componente no lograba la ejecución de sus acciones con los recursos disponibles.

Tabla 34: Número de Reemplazos por problemas de rendimiento por RAM – CPU, y Disco Duro

Componente	Reemplazos - problemas de rendimiento de Memoria- RAM	Reemplazos - problemas de rendimiento CPU	Reemplazos - problemas de rendimiento Disco Duro
A	0	0	0
B	0	0	0
C	0	1	0

La figura 29 muestra el algoritmo de búsqueda invocado por el middleware para buscar y reemplazar el componente C, el cual debe correr bajo el sistema operativo Solaris, y debe ser un componente desarrollado en PHP y con menos de 5 dependencias; ya que no pudo ser mejorado para mantener los

niveles de rendimiento definidos en la configuración inicial del sistema, a través de los ajustes locales que el middleware realizó (ver capítulo 3 y anexo 1 para más detalles de cómo usar el algoritmo de búsqueda).

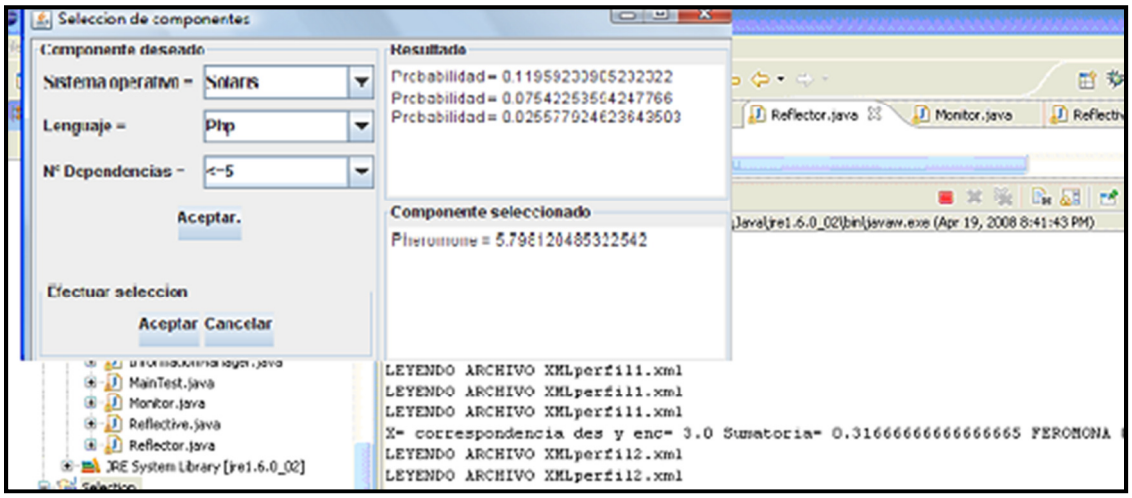

Figura 28: Middleware buscando componentes a reemplazar

En este caso, es importante aclarar que los cambios que realiza el agente Reflector son hechos a nivel del ambiente donde la aplicación se ejecuta, y no a nivel especifico del componente. Dichos cambios son básicamente la liberación de memoria, disco duro o uso de CPU.

Finalmente; y tal como se demuestra en las tablas de resultados, se puede afirmar que cada vez que se simulo baja disponibilidad de memoria, disco duro o CPU, el middleware fue capaz de tomar acciones y liberar los recursos necesarios para mantener la aplicación base en funcionamiento. Este caso de estudio demostró que es posible utilizar el middleware para mantener niveles de rendimiento deseados en una aplicación base.

Capítulo 6

Conclusiones y Recomendaciones

6.1. Conclusiones.

El objetivo general de la tesis fue desarrollar e implementar una arquitectura inteligente de software que facilitara la construcción, el soporte y el monitoreo de aplicaciones basadas en componentes en tiempo de ejecución; para lo cual requería algoritmos de búsqueda y selección de componentes. Dicho objetivo se consiguió a partir de la construcción de un middleware reflexivo que integra agentes inteligentes y algoritmos de selección de componentes de software. Dicho middleware reflexivo propuesto en este trabajo monitorea una aplicación desarrollada utilizando componentes de software, e induce cambios en ella si los objetivos para los que se configuró el middleware son violentados. Además, se lograron los siguientes objetivos específicos:

a. Se propuso un middleware reflexivo, con un módulo específico de búsqueda y selección de componentes, basado en las teorías de computación reflexiva, inteligencia artificial distribuida, arquitecturas de software e ingeniería del software (dominio y componentes).
b. El middleware reflexivo es un modelo de referencia de una Arquitectura Inteligente de Software, que permite monitorear sistemas e integrar componentes reutilizables de software en dichos sistemas, en tiempo de ejecución.

Se crearon dos algoritmos de búsquedas y selección de componentes de software, basados en las teorías de agentes y de inteligencia artificial colectiva, los cuales son ejemplos de implantación artificial de Sistemas Emergentes, una nueva corriente de estudio en el área de Inteligencia Artificial Distribuida [15, 19, 21, 26, 28, 29, 46, 70]; los mismos fueron diseñados, implementados e integrados, no solo como parte del middleware reflexivo, sino como herramientas independientes que ayudan a mejorar el proceso de desarrollo de sistemas, especialmente en la etapa de análisis y selección de componentes

que puedan ser integrados para construir la aplicación.

Así, nuestro trabajo tiene dos aportes fundamentales. En el primer aporte se implementaron dos algoritmos inteligentes, que utilizan aspectos teóricos de inteligencia artificial colectiva para realizar la búsqueda y selección de componentes software en base a probabilidades. Estos algoritmos incorporan la idea de agentes que buscan y seleccionan componentes de software, tal como las hormigas buscan y seleccionan fuentes de alimento. Así, cada vez que un componente es seleccionado o no, el agente incrementa o decrementa el valor de su feromona, según el rendimiento del mismo en un ambiente determinado, lo cual permite que cada vez que se busca un componente la probabilidad de escoger el mejor se incrementa, pues el mismo mantiene un número mayor de feromona en su perfil. Dichas probabilidades fusionan la relación entre los requerimientos deseados (funcionales y no funcionales) de los componentes con los encontrados, con los rendimientos conocidos de dichos componentes encontrados en la plataforma donde uno querrá usarlos. Como se muestra en el capítulo anterior, los resultados obtenidos fueron sumamente interesantes. Así, al comparar estos algoritmos con los mecanismos de búsqueda de los repositorios de componentes, notamos resultados que muestran como nuestros algoritmos generan una selección de componentes mucho más acorde con las necesidades del usuario; lo cual ahorra tiempo y esfuerzo cuando se desarrolla software.

El segundo aporte fue diseñar e implementar un middleware reflexivo, el cual utiliza los algoritmos de búsqueda desarrollados en la tesis como parte de su arquitectura. La idea principal del middleware es tener la posibilidad de monitorear cada uno de los componentes de software que componen una aplicación, a fin de mantener niveles deseados de rendimiento o seguridad en la aplicación; estos últimos fueron los casos de uso utilizados para probar el sistema. Este middleware posee tres agentes principales; uno que funge como Monitor de una aplicación base, otro que toma las decisiones sobre los cambios a generar en la aplicación base, y un agente que gestiona la información y el aprendizaje sobre los cambios y perfiles de los componentes en archivos XML. Como se dijo antes, se realizaron dos casos de prueba para el middleware, uno en el que se configura al middleware para que mantenga los niveles de seguridad de la aplicación base, monitoreando los niveles requeridos de seguridad para los aspectos de autenticación, integridad y confidencialidad en los componentes. El otro caso de prueba se implementó para garantizar el rendimiento de la aplicación base, en cuanto al uso de memoria, CPU, y disco duro por parte de los componentes. El uso del middleware facilita la tarea de monitorear una aplicación base, para mantenerla funcionando según los criterios de eficiencia que se hayan

determinado alcanzar. En ambos casos de prueba, el middleware no solo realizó el monitoreo de la aplicación base, sino que además hizo cambios a sus diferentes componentes, permitiendo mantener los niveles deseados de rendimiento y seguridad de la aplicación base. Cuando fue necesario el reemplazo de algún componente, el middleware reflexivo hizo una llamada al algoritmo B de búsqueda y selección de componentes. Estos resultados demuestran cómo se puede automatizar de manera efectiva, una amplia cantidad de tareas que permanentemente se realizan para mantener sistemas bajo los niveles de rendimiento y seguridad requeridos.

6.2. Recomendaciones.

Tanto el middleware como los algoritmos de selección de componentes de software podrían ser utilizados en muchas otras áreas, como por ejemplo para buscar recursos disponibles en un grid o cluster de procesadores, recursos de internet, servicios web, entre otros. En el caso concreto de nuestros algoritmos de búsqueda y selección, esto es posible debido a que toda la configuración necesaria para usarlos es establecida en archivos XML, independientes de cualquier plataforma o arquitectura. En cuanto al middleware reflexivo, las capacidades de uso del middleware que se presenta en esta tesis son incontables. Aun cuando solo se utilizó para monitoreo de seguridad y de rendimiento, existen muchas otros campos en los que este middleware puede ser utilizado, solo se requiere desarrollar los agentes especializados que desarrollarán las tareas específicas que se requieran en las tareas de gestión de la aplicación base.

Entre los trabajos futuros que se proponen, está la posibilidad de implementar el middleware reflexivo para monitorear no solo a un componente de software como un todo, sino hacer el monitoreo a nivel de los objetos dentro de cada componente; para esto se sugiere la utilización de la biblioteca Proactive (fig. 30). En este caso, cada objeto tiene un meta-objeto asociado al nivel meta-colectivo. Dicho nivel meta-colectivo tendría la tarea de gestionar la aplicación base como un todo, pero interactuando y monitoreando desde el nivel meta-objeto, de esta manera la reflexión se haría tanto en cada objeto como a nivel colectivo. Así, todas la comunicaciones se dan entre el nivel meta-objeto y la aplicación (monitoreo, modificaciones, etc.), a partir de las decisiones que se tomen en el nivel meta-colectivo, con la ayuda de los meta-objetos.

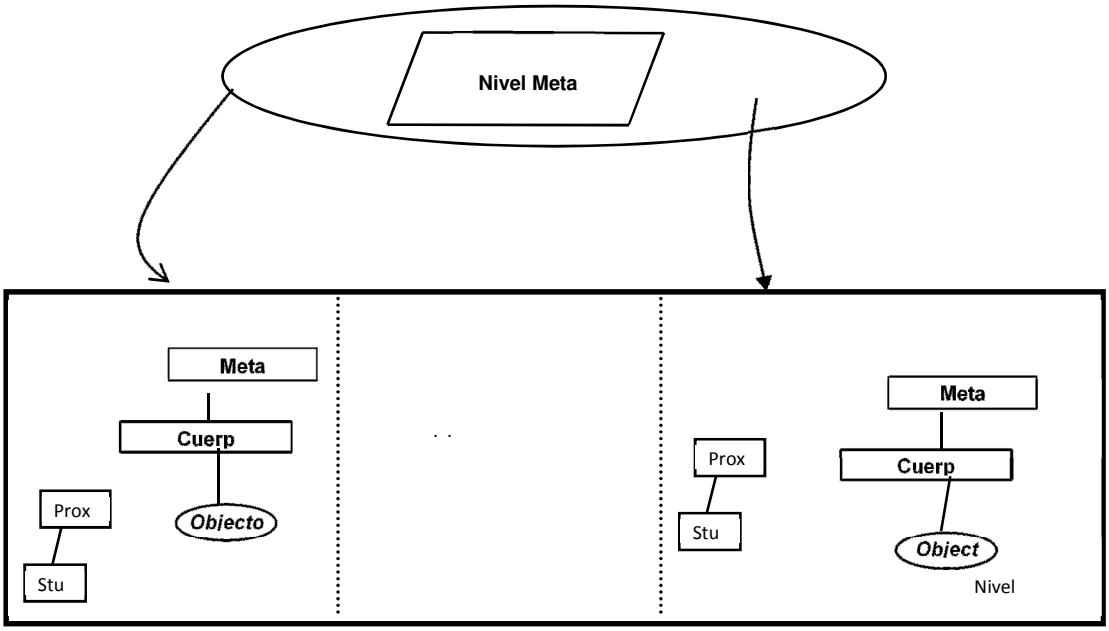

Figura 29: Middleware reflexivo utilizando Proactive como base de modelado

Existe todavía mucho que investigar en áreas como inteligencia artificial, sistemas multiagentes, uso de componentes de software, y computación reflexiva. Así mismo, también existen muchas áreas en las que se puede investigar la utilización del middleware propuesto, diferentes a la de gestión de aplicaciones base, como por ejemplo, para monitorear redes y hacer reflexión sobre los niveles de seguridad y rendimiento de los recursos en ellas. En este caso se estarían haciendo estudios de comportamientos en dichos sistemas, de esta manera se utilizaría el middleware para mantener los registros de comportamientos.

Así mismo, y debido a que los algoritmos AA y AB son independientes del middleware, pueden diseñarse implementaciones particulares para dichos algoritmos o el middleware. Es decir, no necesariamente deben estar atados.

Se recomienda también diseñar un caso de estudio para probar el funcionamiento del middleware en ambientes distribuidos, que utilicen diferentes sistemas operativos o configuraciones; incluso sería interesante evaluar el funcionamiento de una arquitectura que requiera evaluación de seguridad y rendimiento al mismo tiempo, lo cual permitiría incorporar todas las funcionalidades que en esta tesis doctoral se han desarrollado. En el caso de que el Middleware Reflexivo deba ser también distribuido,

debe tomarse en cuenta que todos los agentes que lo conforman deben ser capaces de comunicarse a través del mecanismo indirecto que se implemente, el cual está basado en archivos XML, por tanto los agentes deben tener permiso de lectura y escritura en los directorios donde se mantienen dichos archivos, los cuales deben estar almacenados en un repositorio común y centralizado al que puedan tener acceso todos los agentes. Así mismo, se tendrían los tres agentes del middleware en cada sitio, por lo que en este caso se deben incorporar mecanismos sofisticados de sincronización entre los agentes.

Anexo I

Pre-requisitos:

Ante todo es importante tener en cuenta que se debe cumplir los siguientes pre-requisitos a fin de que el middleware reflexivo, y el algoritmo de búsqueda y selección de componentes, funcionen como se espera:

a. Ambos sistemas deben tener permisos de lectura y escritura en el disco duro donde se corren las aplicaciones.
b. La aplicación a monitorear debe estar desarrollada por componentes de software.
c. Se deben establecer los requerimientos a monitorear y a asegurar en archivos iniciales de configuración, tal como se definen en el punto 2 de este anexo.
d. Debido a que ambos sistemas han sido desarrollados en Java, y aún se encuentran en fase de prototipos, se requiere un compilador JAVA que permita al usuario recompilar las aplicaciones (middleware y selección) cada vez que se le hagan cambios; si estos cambios implican modificación de código fuente.

Archivos Necesarios:

Para la implementación del middleware reflexivo y el algoritmo de selección de componentes, existen tres tipos de archivos que se manipulan, todos están creados en formato XML, los cuales contienen la información tal como sigue:

e. Archivo de configuración inicial (inicial.xml): Se requiere un archivo XML donde se establecen los requerimientos de monitoreo. Es decir; en este archivo se especifican los

nombres de los componentes que componen la aplicación base, los estados que se deben monitorear, ruta de acceso de cada componente, y otras especificaciones según la implementación del middleware (ver ejemplo en la fig 22).

f. Archivo de Estados (status.xml): Es un archivo que guarda el estado de cada componente, para un momento determinado, en este archivo se tienen las variables que se monitorean y los valores actuales de las mismas (ver Fig. 26).

g. Archivos de Componentes (perfil.xml): Cada componente debe tener asociado un archivo XML que identifica su perfil, monto de feromona asociado al mismo, nombre del componente, características etc. (ver fig. 7)

Como Instalar el algoritmo de selección de componentes de software:

h. Se debe crear un directorio (selection) en el Disco Duro donde se copie el ejecutable compilado del algoritmo de selección.

i. Se debe recompilar el algoritmo con los nombres de los repositorios de software donde se desee hacer la búsqueda de componentes. Por ejemplo, sourceforge.net; components.com, etc. Sin embargo, los repositorios que se utilizaron de prueba fueron locales pues se requerían permisos de lectura y escritura para cada perfil de cada componente (Ver fig. 31).

j. El algoritmo debe tener permisos de lectura y escritura en los repositorios de software, estos son permisos a nivel del sistema operativo para poder modificar los archivos XML que corresponden a cada característica y perfiles de los componentes de software (ver figura 32). En este caso se copiaron los componentes y se simularon los repositorios en un servidor disponible para el proyecto.

Como Instalar el Middleware:

k. Se especifica cada cuanto tiempo se desea hacer el monitoreo del sistema base, y se compila la aplicación. Esto puede ser hecho configurando un proceso a nivel del sistema operativo (ver figura 32), o codificado en el middleware dentro del agente Monitor.

l. Se deben programar e incorporar los agentes especializados necesarios para los procesos que se deseen manipular en la aplicación base y recompilar el middleware. Cada agente debe ser una clase JAVA que se integre al proyecto principal y se recompile.

m. Se debe copiar el archivo compilado en el mismo directorio donde se copió el compilado del algoritmo de selección.

n. Se deben hacer las siguientes configuraciones iniciales:

    i. Archivo de Configuración Inicial (inicial.xml): Como se ve en la fig. 22, se deben especificar por lo menos los nombres de los componentes a monitorear, la ruta donde los mismos están instalados, las variables de estado que se desean monitorear, y cuálesserían sus valores ideales.Para el caso de este trabajo en particular se utilizan variables relacionadas a rendimiento y seguridad, que fueron los casos de estudio de esta tesis.

    ii. Se debe crear un archivo de estado vacío, llamado status.xml. (ver figura 26)

    iii. Se debe verificar que cada componente tenga por lo menos un archivo que lo caracteriza. Este archivo se llama perfil.xml. (ver fig .7).

o. Se debe recompilar el middleware luego de hacer cualquier modificación.

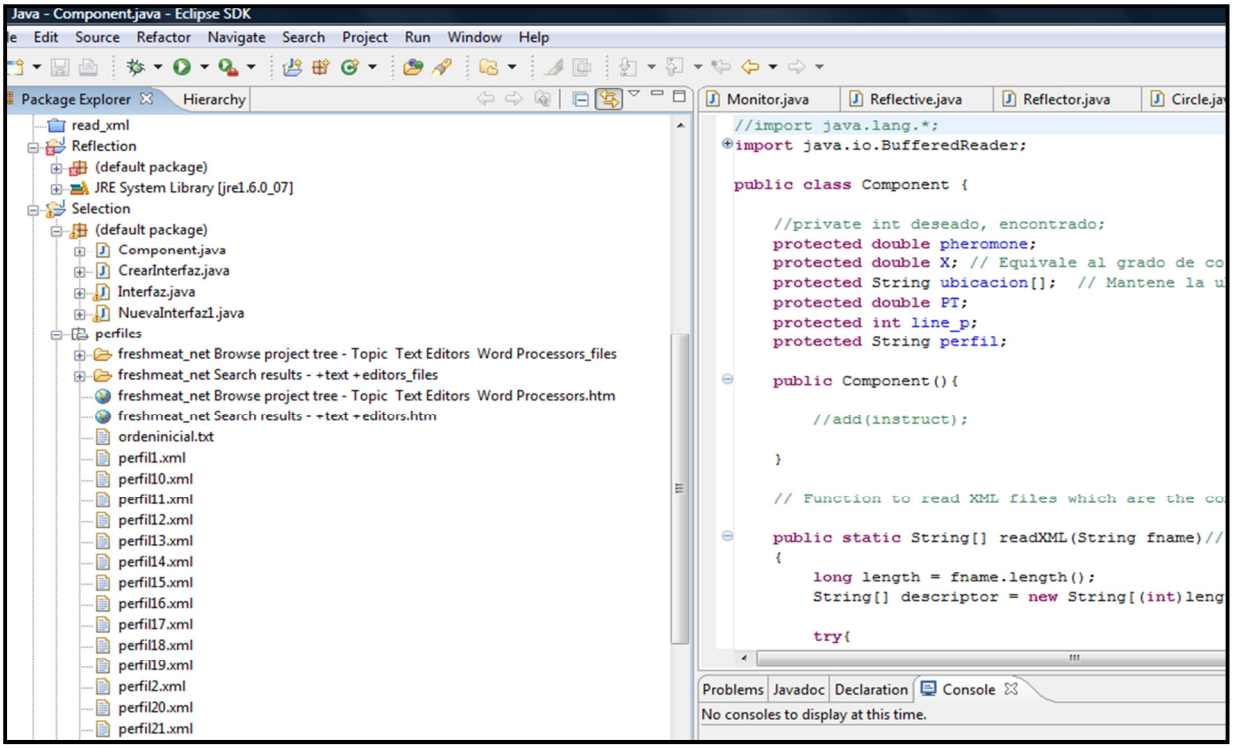

Figura 30: Los repositorios que se utilizaron de prueba fueron locales pues se requiere poder escribir y modificar los perfiles de cada componente

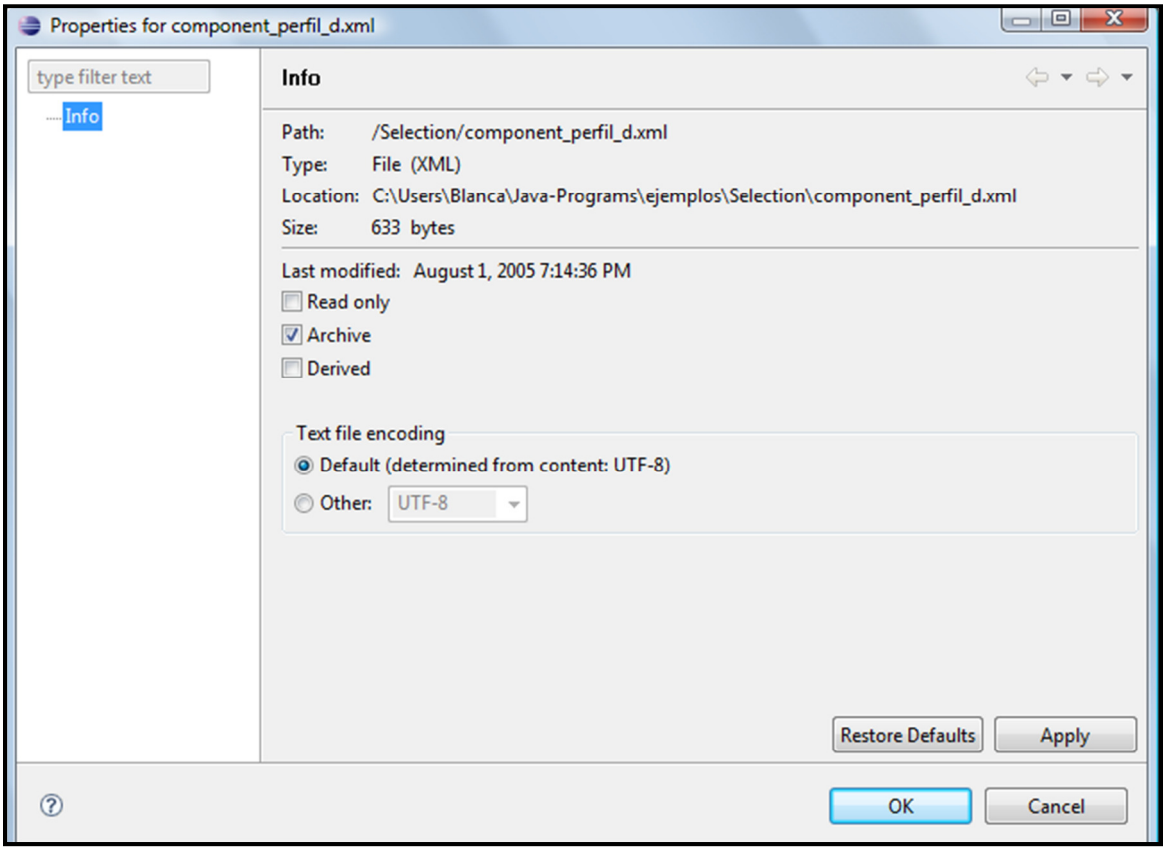

Figura 31: Los archivos XML deben ser de tipo “archive” y deben poder ser modificados (lectura - escritura)

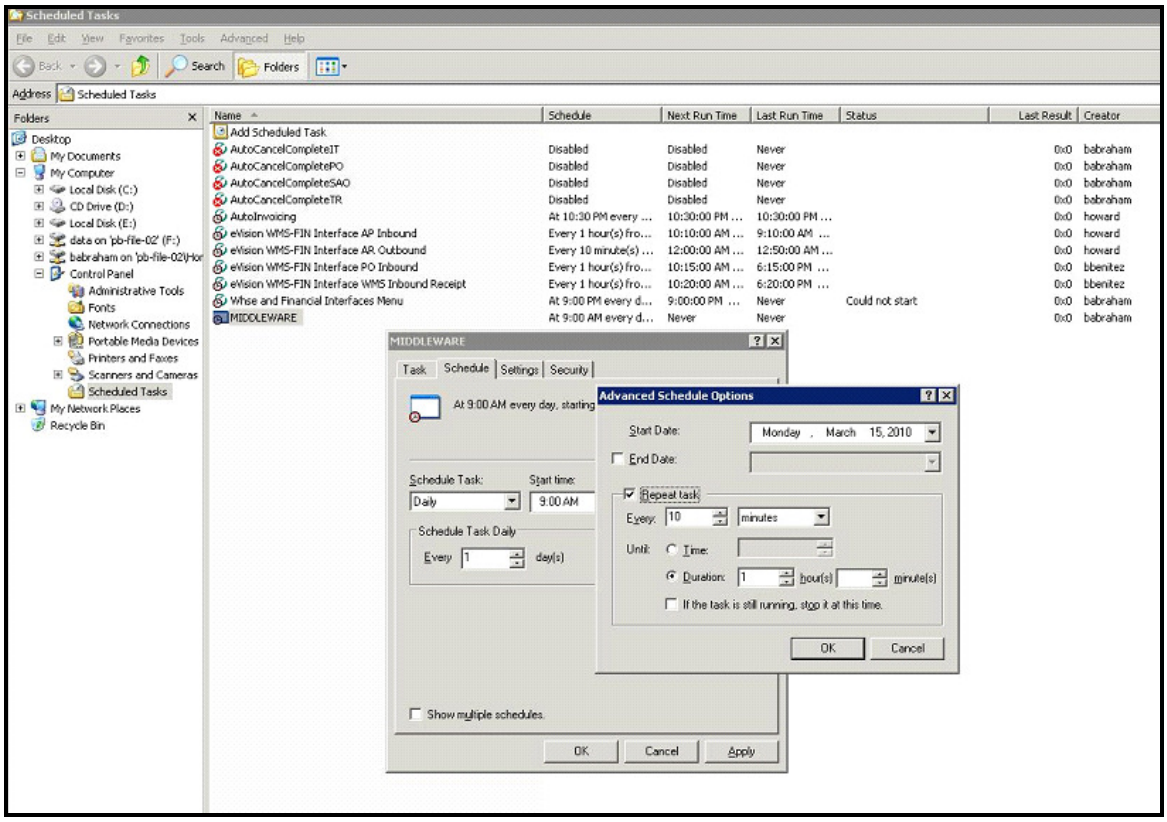

Figura 32: Proceso configurado a nivel de sistema operativo para correr el middleware (monitor) cada 10 minutos

Referencias:

[1] Abraham, B., Aguilar, J., and Leiss, E,. "Middleware for Improving Security in a Component Based Software Architecture". Networked Computing and Advanced Information Management, IEEE Computer Society, 2008. Vol. 1. pp.502-509.

[2] Abraham, B,. Aguilar, J,. Batista, J,. "Selection Algorithm Using Artificial Ant Colonies". WSEA Transactions on Computers, 2006. Vol. 5, No. 10, pp. 2197-2203.

[3] Aguilar, J,. "A General Ant Colony Model to Solve Combinatorial Optimization Problems". Revista Colombiana de Computacion 2001. Vol. 2 No. 1. pp. 7-18.

[4] Aguilar, J,. "The Combinatorial Ant System for Dynamic Combinatorial Optimization Problems". Revista de Matematica: Teoria y Aplicaciones, 2005. Vol. 12, No. 1&2, pp. 51-60.

[5] Aguilar, J,. Abraham, B,. "Sistemas para el Manejo de Incidentes de Seguridad Informática usando Colonias Artificiales de Hormigas", *Revista Gerencia Tecnológica Informática.* Instituto Tecnológico Iberoamericano de Colombia, No. 16, Vol. 6, pp. 41-52, Diciembre 2007.

[6] Aguilar, J,. Abraham, B,. "Software Component Selection Algorithm Using Intelligent Agents", *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Vol. 4496, pp. 82-91, 2007.

[7] Aguilar, J,. Abraham, B,. Moreno, G,. "A security incidents management for a CERT based on Swarm Intelligence", *WSEAS Transactions on Computers*, Vol. 8, No. 8, pp. 1398-1407, August 2009.

[8] Aguilar, J,. Besembel, I,. Cerrada, M,. Hidrobo, F,. Narciso, F,. "Una Metodologia para el Modelado de Sistemas de Ingenieria Orientado a Agentes, Revista Iberoamericana de Inteligencia Artificial, 2008. Vol. 12, No. 38, pp. 39-60. AEPIA (http://erevista.aepia.org)

[9] Aguilar, J,. Velasquez, L,. and Pool, M,. "The Combinatorial Ant System, Applied Artificial Intelligence", Taylor and Francis, 2004. Vol. 18, No. 5, pp. 427-446.

[10] Andreou, A,. Vogiatzis, D,. Papadopoulos, G,. "Intelligent Classification and Retrieval of Software Components". Computer Software and Applications Conference, COMPSAC . Vol. 2, No. 17-21. pp.37 – 40. 2006.

[11] Anurag, M,. Friedman, D,. "Towards a Theory of Reflective Programming Languages". Third Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming, October 1993.

[12] Bachman, F,. et al. "Technical Concepts of Component-Based Software Engineering". Software Engineering Institute, Volume II. Carnegie Mellon Internal Research and Development. 2000.

[13] Barros, T,. Henrio, L,. and Madelaine, E,. "Behavioral Models for Hierarchical Components". Lecture Notes in Computer Science. Springer. 2005, Vol. 3639. pp.154- 160.

[14] Bertoa, M,. Troya, J,. y Vallecillo, A,. "Aspectos de calidad en el desarrollo de software basado en componentes". Universidad de Málaga. 2002

[15] Bonabeau, E,. Dorigo, M,. Theraulaz, G,. "Swarm Intelligence. From Natural to Artificial Systems". Oxford University Press. 1999.

[16] Braga, R,. Werner, C,. and Mattoso, M,. "Odyssey-search: An agent system for component". Journal of

Systems and Software. 2006. Vol 79. No. 2. pp. 204-205.

[17] Brown, A,. and Wallnau, K,. The current state of CBSE. IEEE Software, 1998. Vol. 15, pp. 37-46

[18] Cai, W,. Coulson, G,. Grace, P,. et al. "The Gridkit Distributed Resource Management Framework". EGC 2005. pp. 786-795

[19] Chalmers, D,. "Strong and Weak Emergence" 2002. http://consc.net/papers/emergence.pdf Republished in P. Clayton and P. Davies, eds. "Fuertes y débiles Emergence" http://consc.net/papers/emergence.pdf Reeditado en Clayton, P. y P. Davies, eds. (2006) The Re-Emergence of Emergence . la reaparición de la emergencia. Oxford: Oxford University Press. 2008. Oxford: Oxford University Press.

[20] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., "Non-Functional Requirements in Software Engineering". Kluwer Academic Publisher, 2000.

[21] Colorni, A,. Dorigo, M,. Maniezzo, V,. "Distributed Optimization by Ant Colonies". actes de la première conférence européenne sur la vie artificielle, Paris, France, Elsevier Publishing, 1991. pp. 134-142.

[22] Componentes, http://www.monografias.com/trabajos16/componentes/ componentes.html Revisado - Agosto 2007.

[23] ComponentSource. www.componentsource.com Revisada en Octubre 2005.

[24] Corsava, S,. Getov, V,. "Intelligent Architecture for Automatic Resource Allocation in Computer Clusters". International Parallel and Distributed Processing Symposium (IPDPS'03), 2003. pp.201a.

[25] Coulson, G,. Grace, P,. Blair, G,. et al. "A Component-based Middleware Framework for Configurable and Reconfigurable Grid Computing ". Concurrency and Computation: Practice and Experience, 2006.

[26] De Wolf, T,. Holvoet, T,. "Emergencia Contra Auto-organización: conceptos diferentes pero prometedor cuando se combina", Ingeniería de Sistemas autónomos Organizador: metodologías y aplicaciones, Lecture Notes in Computer Science 2005, Vol. pp. 34-64.

[27] Dorigo, M,. and Stutzle, T,. "Ant Colony Optimization". Branford Books. 2004.

[28] Dorigo, M,. Gambardella, L,. "Ant colonies for the traveling salesman problem". BioSystems 1997. Vol. 43 pp. 73-81.

[29] Dorigo, M,. Maniezzo, V,. Colorni, A,. "The ant system: optimization by a colony of cooperating agent". IEEE Transactions on System, Man, and Cybernetics—1996. Part B. Vol. 1. pp.29-42.

[30] Ferber, J,. "Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence". Addison-Wesley. 1999.

[31] Forman, I,. Forman, N,. "JAVA Reflection in Action". Manning. 2005.

[32] Francoise, B,. Denis, C,. Morel, M,. "From Distributed Objects to Hierarchical Grid Components". International Symposium on Distributed Objects and Applications (DOA), November 2003.

[33] Francoise, B,. Denis, C,. Morel, M,. "On Hierarchical, Parallel and Distributed Components for Grid Programming". Workshop on Component Models and Systems for Grid Applications, June 2004.

[34] Gokhale, A,. Schmidt, D,. "Techniques for optimizing CORBA middleware for distributed embeddedsystems". INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and

Communications Societies. Proceedings. IEEE. Mar 1999. Vol. 2, pp. 513-521.

[35] Gokhale,A,. Cross,B,. Cross,R,. Schmidt,D,. "Towards Real-time Fault-Tolerant CORBA Middleware". Cluster Computing 2004. vVol.7. No.4 pp. 331-346

[36] Gonzalez de Miguel, A,. "The Design of an Intelligent Software Architecture Based on a Generic eLearning Environment". Intelligent Tutoring Systems. 8th International Conference, ITS 2006, Jhongli, Taiwan, Springer, Lecture Notes in Computer Science, Vol. 4053, pp. 756-759

[37] González, A,. Mijares, M,. Mendoza, L,. Grimán, A,. Pérez, M,. "Método de Evaluación de Arquitecturas de Software Basadas en Componentes". (MECABIC). VII Jornadas Internacionales de las Ciencias Computacionales, Colima, México. 2005

[38] Grimm, R,. Davis, J,. Lemar, E,. MacBeth, A,. et al. "System support for pervasive applications". ACM Transactions on Computer Systems, 2004; pp. 421-486

[39] Grundy, J,. "An implementation architecture for aspect – oriented component engineering". Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2000, Las Vegas, Nevada, USA. CSREA Press 2000.

[40] Heineman, G,. and Councill, W,. et al., "Component-Based Software Engineering: Putting the Pieces Together". Addison-Wesley, 2001.

[41] Heineman, G,. Council, W,. Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley Professional. 2004.

[42] Herzum, P,. and Sims, O,. "Business component factory : A comprehensive overview of component-based development for the enterprise". John Wiley & Sons, 2000.

[43] http://freshmeat.net . Revisado Julio 2006.

[44] http://sourceforge.net. Revisado Julio 2006.

[45] http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf. Revisado Noviembre 2009.

[46] Kolp, M,. "A Metaobject Protocol for Integrating Full-Fledged Relationships into Reective Systems". Thesis submitted for the degree of Doctor of Philosophy Department of Information and Documentation Sciences Universit_e libre de Bruxelles – ULB. Octubre 1999.

[47] Kon, F,. Roman, M,. Liu, P,. Mao, J,. Yamane, T,. Magalhes, L,. Campbell R,. "Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB". IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000). 2000. pp.121-143.

[48] Licata, I,. y Sakaji, A,. (eds.). Física de la emergencia y de la Organización, 2008. ISBN 13 978-981-277-994-6, Mundial de la Ciencia y el Imperial College Press.

[49] Lindemann, J,. Dahlblom, O,. Sandberg, G,. "Using CORBA Middleware in Finite Element Software". Lecture Notes in Computer Science Springer Berlin / Heidelberg. 2002. Vol. 2331/2009. pp. 701-710.

[50] Liu, Y,. Passino, K,. "Swarm Intelligence: Literature Overview". Department of Electrical Engineering. The Ohio State University. Literature Overview. March 2000.

[51] Majumdar, S,. Shen, E,. Abdul-Fatah, I,. "Performance of adaptive CORBA middleware". Journal of Parallel and Distributed Computing, 2004. Vol.64 No.2, pp.201-218.

[52] Medina, J,. Drake, J,. “Modelado y Análisis de Tiempo Real de Aplicaciones Basadas en Componentes” V Jornadas de Tiempo Real. Cartagena, 2002.

[53] Méndez, F,. "Towards a standardization of multi-agent system frameworks". Computer Science Department, University of Calgary, Canada. 1999.

[54] Nils, J,. Nilsson, J,. "Inteligencia Artificial una nueva síntesis". McGrawHill. Primera versión en español, 2001.

[55] Objectweb. FRACTAL Project. http://fractal.objectweb.org/tutorial/index.html. Revisado - Agosto 2008

[56] Othman, O,. O'Ryan, C,. Schmidt, D,. "CORBA Middleware-Based Load Balancing". IEEE Online. 2001.Vol 2. No 3.

[57] Parashar, M., Harir, S., "Autonomic Computing, Concepts, Infrastructure and Applications". CRC Press, 2007 by Taylor and Francis Group.

[58] Pilone, D,. Pitman, N,. "UML 2.0 in a Nutshell". O'Reilly. 2005.

[59] Pressman, R,. "Ingeniería del Software. Un Enfoque Práctico". McGrawHill, Cuarta Ediciòn. 1997.

[60] Reina, A,. Torres, J,. "Components + Aspects: A General Overview". Revista Colombiana de Computación. 2004. N.1 Vol. 5. pp. 77-95.

[61] Reinoso, B,. "Introducción a la Arquitectura de Software". Versión 1.0. Universidad de Buenos Aires, Argentina. Marzo 2004.

[62] Rich, E,. Knight, K,. "Artificial Intelligence". McGraw Hill, Traducción española: Inteligencia Artificial. Segunda Edición. McGraw-Hill, 1994.

[63] Román, M,. Hess, C,. Cerqueira, R,. Ranganathan, A,. Campbell, R,. and Nahrstedt, K,. "Gaia: A Middleware Infrastructure to Enable Active Spaces". IEEE Pervasive Computing, 2002. pp. 74-83.

[64] Rosa, N,. Alves, C,. Cunha, P,. Castro, J,. Justo, G,. "Using Non-Functional Requirements to Select Components: A Formal Approach". In Proceedings of IDEAS’01, San José, Costa Rica. April 2001.

[65] Roshandel, R,. and Medvidodic, N,. "Modeling Multiple Aspects of Software Components". Proceeding of Workshop on Specification and Verification of Component-Based System. 2003. Helsinki, Finland, , ESEC-FSE03. pp. 88-91

[66] Russell, S,. Norvig, P,. "Artificial Intelligence: A Modern Approach". 2nd ed. Prentice Hall 2003.

[67] Seacord, R,. Hissam, S,. Wallnau, K,. "Agora: A Search Engine for Software Components". Technical Report CMU/SEI-98-TR-011. 1998.

[68] Silvestri, F,. Puppin, D,. Laforenza, D,. Orlando, S,. "Concurrency and Computation". Practice & Experience archive. 2006. Vol. 18, No. 10. pp. 1317 – 1331.

[69] Smith, B,. "Reflection and Semantics in Lisp". Conference Record of the 14th Annual ACM Symposium on Principles of Programming Languages, Salt Lake City, Utah. 1984.

[70] Socha, K,. and Dorigo, M,. "Ant Colony Optimization for Continuous Domains”. European Journal of

Operational Research, March 2008, Vol. 185, No. 3, pp. 1155-1173

[71] Souza, R., Costa, M., Braga, R,. et al. "Software components retrieval through mediators and web search". J. Braz. Comp. Soc., Nov. 2002, Vol.8, No.2, pp.55-63.

[72] Stajano, F,. "Security for Ubiquitous Computing". Wiley Series in Communications Networking & Distributed Systems. 2002.

[73] Szyperski, C,. "Component Software: Beyond Object-Oriented Programming". Addison- Wesley, 1999.

[74] Troelsen, A,. "Pro C# with .NET 3.0". Special Edition. Apress. 2007.

[75] Vivancos, E,. Hernández, L,. Botti,V,. "Inteligencia Artificial Distribuida en entornos de tiempo real". Revista Iberoamericana de Inteligencia Artificial 1998. Vol. 6 pp. 24-35.

[76] www.wikipedia.com, búsqueda “Inteligencia Colectiva” – Revisión 30 Septiembre 2007.

[77] www.wikipedia.com, búsqueda “Sistemas Multiagentes” – Revisión 30 Septiembre 2007.

[78] Xu, J,. Randell, B,. and Zorzo, A,. "Implementing Software-Fault Tolerance in C++ and Open C++: An Object-Oriented and Reflective Approach". Proc. CADTED'96, Bejing, China, 1996. pp. 224-229.