UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
POSTGRADO EN COMPUTACIÓN

Diseño de un Sistema Manejador de Recursos para un Sistema Operativo Web

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
POSTGRADO EN COMPUTACIÓN

Diseño de un Sistema Manejador de Recursos para un Sistema Operativo Web

Tesis presentada por el
Lic. Edgar Ferrer

para optar al título de
Magíster en Computación

Tutor:
PhD. José Aguilar

Mérida, Julio de 2003.

dedicado a Merle y Hugo

Diseño de un Sistema Manejador de Recursos para un Sistema Operativo Web

Por Edgar Ferrer
Bajo la Asesoría del Dr. José Aguilar

Resumen

En los últimos años la Web ha experimentado un crecimiento acelerado, se han introducido nuevas funcionalidades y tecnologías y la web ha dejado de ser solamente una inmensa fuente de información para convertirse en una inmensa fuente de recursos computacionales. Ha trascendido la idea de crear sistemas que permitan administrar recursos computacionales en la web, surge así la noción de Sistema Operativo Web (SOW).

El presente trabajo se encuentra enmarcado dentro de un proyecto de diseño de un Sistema Operativo Web, en él se manejan aspectos relacionados con el diseño de un Sistema Manejador de Recursos para un SOW, el cual está concebido bajo un enfoque de sistemas multiagentes.

El Sistema Manejador de Recursos propuesto se encarga de realizar actividades de localización, configuración y asignación de recursos computacionales abordando problemas propios del manejo de recursos en la web como lo son la volatilidad y heterogeneidad de los recursos computacionales. El diseño presentado se apoya en una metodología para el desarrollo de sistemas multiagentes conocida como MAS-*Common*KADS y se han implementado simulaciones del sistema diseñado con el fin de evaluar su desempeño.

Agradecimientos

Quiero agradecer al Dr. José Aguilar su asesoría en este trabajo. Gracias a su constante guía y a sus recomendaciones, este trabajo es hoy una realidad.

Mis agradeciminetos a los Ings. Niraska Perozo y Juan Vizcarrondo, compañeros del proyecto "Sistema Operativo Web", por sus contribuciones.

Mi agradecimiento a los estudiantes del Postgrado en Computación de la Universidad de los Andes, en especial a los Ings. Manuel Correa y Hermes Díaz por todo el apoyo brindado y por su amistad. Agradezco también a la secretaria del Postgrado en Computación Luisa Díaz Figuera por su incondicional colaboración en los momentos más necesarios. Quiero agradecer también a Analí Silva y a Alexis Valero por el gran apoyo brindado, y un especial agradecimiento a Maria Alexandra por apoyarme en todo momento, su valiosa colaboración aparece reflejada en la páginas de este trabajo.

Agradezco al Dr. Jaime Seguel y a la Dra. Dorothy Bollman de la Universidad de Puerto Rico por sus valiosos aportes. Agradezco también a los amigos de La Universidad del Zulia y la Universidad Simón Bolívar. Finalmente, un reconocimiento muy especial a nuestra *alma mater* la Universidad de los Andes, para esta institución la mayor gratitud de mi parte.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Generalidades

En años recientes la *Word Wide Web* ha experimentado un crecimiento vertiginoso estimulado por el desarrollo de nuevas aplicaciones y tecnologías basadas en la Web. Inicialmente, el principal atractivo de la Web era el fácil acceso a una gran cantidad de información distribuida alrededor del mundo, posteriormente se comenzó a explotar la capacidad de ejecutar Aplicaciones Web. Se desarrolló entonces un modelo de Computación Web con las siguientes capacidades: la información residente en un *host* remoto puede ser bajada automáticamente y vista en una máquina cliente, y las aplicaciones residentes en un *host* remoto pueden ser bajadas automáticamente y ejecutadas en una máquina cliente. Este modelo ha sido muy exitoso y ha contribuido al desarrollo de campos como el comercio electrónico y las aplicaciones multimedia. Sin embargo, dicho modelo sólo explota una fracción de todo el potencial de Internet. Un enfoque más ambicioso es ver la Web no sólo como una inmensa fuente de información, sino también como una inmensa fuente de recursos computacionales.

Recientemente ha surgido el interés de explotar los recursos computacionales de la Web y se han desarrollado nuevas tecnologías y conceptos en el campo de las redes y la computación móvil, como lo son la computación grid [19], la meta-computación [14], y la computación P2P (*peer to peer*)[36]. También se ha comenzado a manifestar la necesidad de disponer de sistemas operativos que permitan compartir recursos computacionales y distribuir servicios entre los nodos de la Web [41]. Particularmente, ha trascendido la idea del desarrollo de sistemas operativos para la Internet, es decir, que el *World Wide Web (WWW)* pudiera ser

soportado y manejado por un gran sistema operativo virtual, al cual se ha denominado Sistema Operativo Web (SOW).

El objetivo de un SOW es proveer una plataforma que permita a los usuarios beneficiarse de los recursos computacionales ofrecidos por la Web. Esto es, hacer disponible a todos los sitios de la red recursos que permitan ejecutar tareas o aplicaciones para las cuales los recursos locales no son suficientes [41].

En esencia, un SOW debería cumplir con las mismas funciones de un sistema operativo convencional, pero además, debe considerar algunos problemas que surgen cuando el dominio es el *WWW*. Aspectos tales como la portabilidad y la seguridad, son a considerar. Sin embargo, la **naturaleza heterogénea** y la **rápida evolución** de la *Web* son los dos problemas principales [29], los cuales no pueden ser tratados como dos problemas aislados. Por ejemplo, el problema de la **heterogeneidad** de las redes ha sido tratado particularmente en los trabajos de meta-computación, pero la Web es más que un simple meta-computador, ya que para la Web no se puede tener un catálogo de todos los recursos disponibles debido a la **rápida evolución** de los mismos.

Teniendo en cuenta las consideraciones anteriores, debemos idear mecanismos que permitan proveer los servicios propios de un sistema operativo, y que a la vez, estos mecanismos se adecuen a los rasgos especiales de la Web. El SOW debe proveer servicios a los usuarios, tales como:

Gestión de memoria.

Seguridad y accesibilidad.

Comunicaciones.

Planificación de procesos y balance de carga.

Gestión de archivos y de Entradas/Salidas.

Tolerancia a falla, etc.

Debido a la naturaleza dinámica de la Web, la lista de los servicios que debe proveer el SOW podría extenderse con el tiempo, es decir, que en el futuro podrían surgir nuevas necesidades y servicios actualmente inimaginables. En virtud de esta situación, la cantidad y

la diversidad de servicios sería tal, que el poder concebir un sistema operativo que dé soporte para cada uno de los servicios se convertiría una tarea algo difícil. Problemas como este son abordados en el presente trabajo, donde estudiamos aspectos relacionados con el manejo de recursos en un SOW. Proponemos un diseño de Sistema Manejador de Recursos (SMR) basado en el uso de motores de búsqueda, los cuales emplean mecanismos de razonamiento para la localización y configuración de recursos.

1.2. Motivación

El manejo de recursos es una actividad clásica en todo sistema operativo, por lo tanto, un componente fundamental de los sistemas operativos es el Sistema Manejador de Recursos. Un sistema manejador de recursos debe estar en capacidad de [14]:

1. Localizar y asignar recursos computacionales.

2. Efectuar actividades que permitan preparar un recurso para ser usado.

El concepto anterior aplica, en general, a los sistemas operativos. Si deseáramos adaptar dicha idea al caso particular de los SOW, tendríamos que afrontar los siguientes problemas (ver figura 1.1):

1. *Volatilidad de los recursos.* Debido a la naturaleza dinámica de la Web, no se puede tener un catálogo con todos los recursos disponibles. En un ambiente Web un recurso puede cambiar de sitio, o un sitio Web puede permanecer temporalmente fuera de línea, o incluso se pueden agregar nuevos servicios en la Web. Situaciones como éstas dificultan la tarea de **localizar recursos computacionales** en la Web.

2. *Heterogeneidad de los recursos.* La Web no solo es heterogénea a nivel conceptual (por la diversidad de servicios que ofrece), sino que también lo es a nivel físico, es decir, a nivel del hardware y software disponible en diferentes sitios de la red. Se pueden encontrar diversas tecnologías de red y diferentes máquinas como computadores personales, estaciones de trabajos, supercomputadores, etc. Cada una de ellas con sus particulares configuraciones a nivel de sistemas operativos, protocolos y aplicaciones. Como conse-

cuencia de lo anterior, podríamos tener un mismo recurso con distintas configuraciones o versiones, este hecho dificulta la tarea de **preparar un recurso para ser usado.**

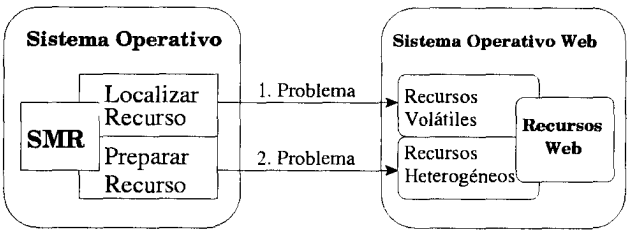

Figura 1.1: Problemas a considerar en el diseño de un SMR para un SOW

En este trabajo se propone atacar el problema de la heterogeneidad aplicando la idea de un diseño "versionado" del SOW [30]. Esto es, tener diferentes versiones de servicios corriendo simultáneamente en diferentes nodos de la red, y proveer al SOW de mecanismos que permitan determinar cual es la versión más adecuada para satisfacer un requerimiento determinado. Por otro lado, se propone atacar el problema de la volatilidad de recursos haciendo uso de motores de búsqueda que permiten procesar inteligentemente los requerimientos del SOW.

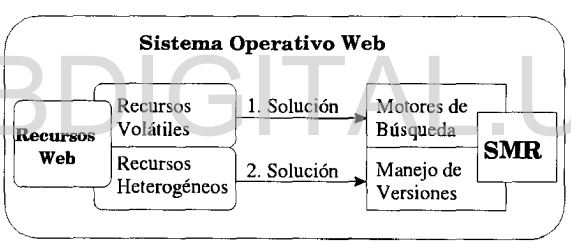

Figura 1.2: Soluciones a los problemas de manejo de recursos en un SOW

Con las soluciones propuestas anteriormente se afrontan dos importantes retos que deben ser considerados para la eficiente implementación de un gran sistema operativo como lo es el SOW, éstos son: la escalabilidad y la adaptabilidad del sistema. El SMR debe establecer los mecanismos propios del manejo de recursos dentro de un ambiente escalable y adaptable. Es difícil utilizar técnicas predefinidas que puedan tratar con la escalabilidad y adaptabilidad del sistema simultáneamente, para eso se sugiere utilizar métodos dinámicos y mecanismos de inteligencia para construir un sistema robusto donde todos sus componentes asumen sus propios roles y cooperan entre si para la solución de problemas. Lo anterior nos lleva a considerar una metodología basada en agentes como una estrategia adecuada para el desarrollo

del SOW, razón por la cual proponemos en este trabajo un desarrollo bajo un enfoque de Sistemas Multiagente.

1.3. Trabajos Relacionados

Existen varias propuestas para el manejo e integración de los recursos computacionales disponibles en un sistema de computación global. A continuación presentamos varias de esas propuestas.

1.3.1. WOS^{TM}

El proyecto Web Operating System, o WOS^{TM} [27], tiene rasgos comunes con nuestra propuesta. WOS^{TM} Permite el manejo e integración de los recursos tratando el problema de la heterogeneidad y volatilidad en la Web. Este proyecto, al igual que nuestra propuesta, está basados en la idea del uso de versiones como solución a esos problemas. La arquitectura del sistema WOS^{TM} se presenta en la figura 1.3. La ejecución de un servicio en el WOS se inicia en la unidad de.Control de Recursos del Usuario (User Resource Control), quien decide si el servicio debe ser ejecutado localmente o no. Si no puede ser ejecutado localmente, se inicia otra búsqueda controlada por la Unidad de Recursos Remotos (Remote Resource Control) a través de la cual se explora la red hasta encontrar el servicio deseado, luego el mismo es asignado para iniciar la ejecución. En este punto, toda la coordinación del intercambio de los datos entre el usuario y la aplicación será controlada por la unidad de control de tareas (Job Control Unit) hasta que la ejecución del servicio finalice.

Aunque existen afinidades conceptuales entre WOS^{TM} y nuestra propuesta de SOW, hay marcadas diferencias en aspectos fundamentales como lo son el modelo de diseño, la arquitectura y el modelo de comunicación. En cuanto al modelo de diseño, nuestra propuesta está enfocada en un modelo basado en Sistemas Multiagente, donde se destacan los siguientes subsistemas: (1) Subsistema Manejador de Recursos, (2) Subsistema Manejador de Repositorios, (3) Subsistema Manejador de Comunidades, y (4) Subsistema Manejador de Objetos Web; mientras que el WOS^{TM} está basado en un sistema modular constituido por cuatro componentes fundamentales: (1) Interfaz de Usuario (UI), (2) Unidad de Control de Recursos (RCU), (3) Unidad de Control de Recursos Remotos (RRCU), y (4) la capa de Comuni-

cación (WOSCL). La arquitectura del WOS^{TM} está estructurada en tres capas (ver figura 1.3): Repositorio (*Warehouse*), Motor de Búsqueda (*Eduction and Search Engine*), y el Administrador de Servidores y Usuarios (*Host Machine Manager / User Manager*); en cambio, nuestro SOW posee una arquitectura constituida por componentes funcionales o subsistemas cuyo funcionamiento está determinado por agentes autónomos que interactúan entre sí. Las comunicaciones en el WOS^{TM} son manejadas en dos niveles, el primer nivel, es un protocolo genérico del WOS^{TM} conocido como WOSP (WOS Protocol), el cual es usado para ejecutar los servicios y transmitir los resultados de la ejecución, el segundo nivel, es un protocolo conocido como WOSRP (WOS Request Protocol), el cual es usado para localizar o descubrir servicios. En nuestra propuesta de SOW las comunicaciones son manejadas a través de esquemas de comunicación y coordinación entre los agentes del sistema, en relación al WOS^{TM}, nuestro SOW introducen nuevas ideas que ayudan a mejorar las comunicaciones, como lo son: las búsquedas inteligentes, el manejo de comunidades de nodos, y la a migración y replicación de Objetos Web mediante mecanismos de computación emergente.

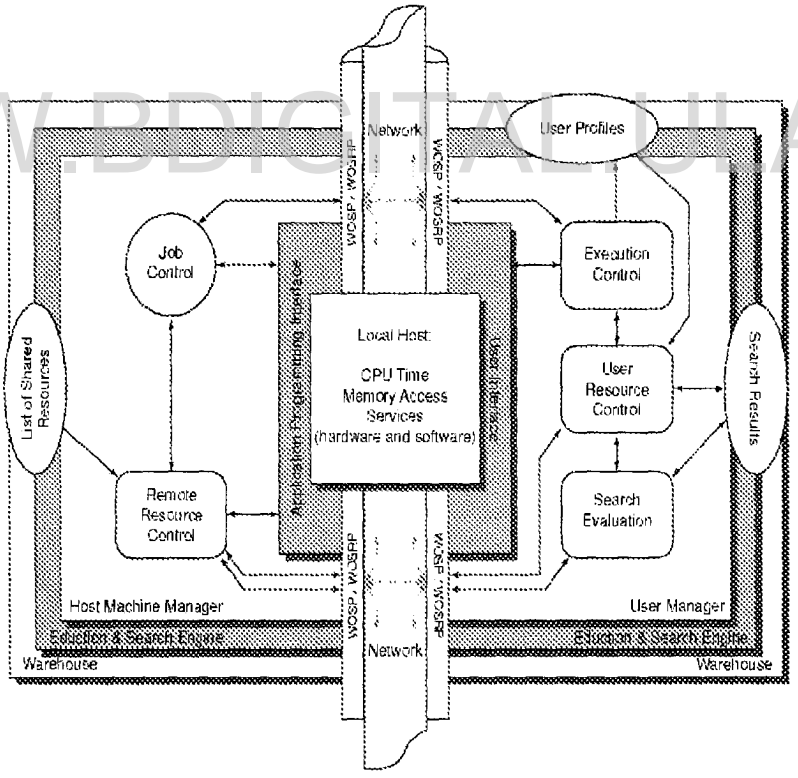

Figura 1.3: Arquitectura del Sistema WOS^{TM} (Tomado de [28])

1.3.2. Netsolve

Otro esfuerzo para explotar los recursos distribuidos en computación global es Netsolve, el cual es un sistema de cómputos distribuido cuyo objetivo principal es ofrecer funcionalidades útiles a un gran abanico de aplicaciones científicas. NetSolve está siendo desarrollado por la Universidad de Tenessee [1], el sistema provee acceso a recursos computacionales, tanto de software como de hardware. NetSolve está escrito en C y las funciones de su API pueden ser llamadas desde Fortran 77 o C. El sistema de comunicación de Netsolve está construido a partir de la biblioteca de los sockets para computadores heterogéneos no fuertemente acoplados. La figura 1.4 muestra la arquitectura de NetSolve. Se pueden observar tres componentes principales en dicha arquitectura: los clientes de NetSlove, el agente NetSolve y los servidores NetSolve. Los clientes NetSolve son escritos por los usuarios y enlazados con la biblioteca de NetSolve. Estas aplicaciones hacen llamadas a servicios específicos de NetSolve. A través de la API, los usuarios de NetSolve acceden a recursos computacionales distribuidos sin que se necesite saber algo de redes de computadores o sistemas distribuidos. De hecho, los usuarios ni siquiera necesitan saber que sus programas invocan recursos remotos.

El agente NetSolve mantiene una base de datos de los servidores NetSolve, con sus capacidades y estadísticas de uso. El agente utiliza esta información para asignar servidores de recursos a los pedidos de los clientes NetSolve. El agente encuentra el servidor que atenderá el pedido lo más rápido posible y lleva una lista de los servidores que tienen problemas.

Los servidores NetSolve son demonios que atienden los pedidos de los clientes. Los servidores pueden estar corriendo en estaciones de trabajo, clusters de estaciones de trabajo o computadores masivamente paralelos.

Un requerimiento de un cliente NetSolve se ejecuta como sigue. El cliente pide al agente NetSolve que se le asigne un servidor NetSolve. El cliente se pone en contacto con el servidor asignado y le manda los parámetros de entrada. El servidor atiende el requerimiento al correr el servicio apropiado antes de devolver el resultado o un error al cliente.

NetSolve permite solamente la ejecución local de una aplicación secuencial que invoque servicios remotos, como por ejemplo, una función matemática o en paralelo. En otras palabras, un usuario de NetSolve no puede desarrollar ni programas secuenciales para ser ejecutados

[1] http://www.cs.utk.edu/netsolve

remotamente ni programas paralelos con llamadas explicitas a una biblioteca de pase de mensajes, si no que desarrolla programas secuenciales que son ejecutados localmente y que invocan servicios remotos en forma transparente.

En NetSolve, el ambiente de comunicación de alta interoperabilidad es la biblioteca de los sockets y no dispone de una biblioteca de paso de mensajes de alto desempeño, lo que no permite la ejecución de programas paralelos. Los clientes, el agente y los servidores.

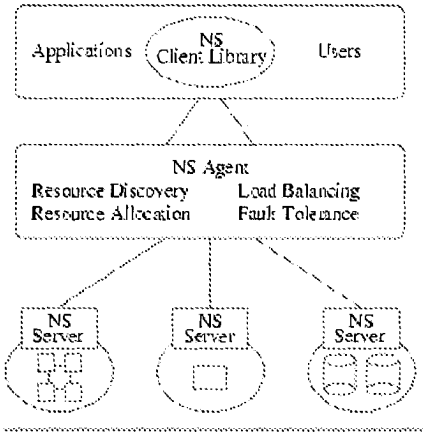

Figura 1.4: Arquitectura de Netsolve (Tomado de [11])

1.3.3. Legion

Legion [20] es un metasistema orientado a objetos que permite interconectar un conjunto de computadores heterogéneos y geográficamente distribuidos, ofreciendo una máquina virtual coherente y simple. El sistema está organizado en clases y metaclases; tanto el hardware como el software que comprenden el sistema están representados por objetos. Los objetos de Legion pueden estar activos o inertes. Un objeto activo corre como un proceso y está siempre dispuesto a atender una invocación a uno de sus métodos. Un objeto inerte es la imagen de un objeto la cual está guardada en un dispositivo de almacenamiento. Esta imagen contiene suficiente información para que el proceso sea reactivado cuando sea necesario. Legion provee un mecanismo de comunicaciones seguras basado en el cifrado de claves públicas y privadas. Los objetos pueden utilizar la clave pública del objeto destino para cifrar los mensajes, el destinatario utiliza su clave privada para descifrar los mensajes que le fueron enviados y que

fueron cifrados con su clave pública.

Legion presenta una arquitectura por capas (ver figura 1.5) la cual consta de cinco niveles: (1) la infraestructura, (2), los servicios de administración de los objetos, (3) los servicios de alto nivel (sistema de archivos, servicios de directorios de nombre, servicios de administración de recursos), (4) la biblioteca Legión (servicios de inacción de métodos), y (5) las aplicaciones de usuarios.

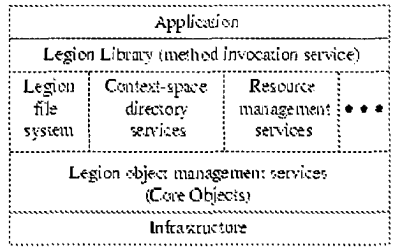

Figura 1.5: Arquitectura de Legion (Tomado de [20])

1.3.4. Globus

Globus [18] proporciona una infraestructura de software que permite a las aplicaciones manejar recursos distribuidos y heterogéneos como si fuera un solo computador. Globus construye una rejilla (grid) computacional que permite el acceso a los recursos computacionales independientes de la posición geográfica.

La arquitectura por capas de Globus se encuentra en la figura 1.6 y consta de cuatro niveles: (l) la estructura de la rejilla,(2) los servicios de la rejilla, (3) los *toolkits* para aplicaciones y (4) las aplicaciones.

La **estructura de la rejilla** está compuesta por los computadores, el sistema operativo, la red de interconexión, los dispositivos de almacenamiento, los enrutadores , etc. Esta capa corresponde a la capa de hardware y sistemas operativos de nuestra arquitectura para la construcción de metasistemas.

Los **servicios de rejillas** integran los diferentes componentes de la rejilla y proveen servicios como GRAM (Globus Resources Allocation Manager) y GRIS (Grid Information Service), entre otros. GRAM en una biblioteca básica que provee los servicios para la ejecución remota. GRIS es una biblioteca con funciones para obtener información sobre el metasistema.

Por ejemplo, se puede preguntar a GRIS información sobre los computadores (¿números de procesadores disponibles?), los dispositivos de almacenamiento (¿es una cinta o un disco duro?) o la red que se quiere utilizar (¿ancho de banda de la red?). Esta capa de Globus incluye una biblioteca de comunicación de bajo nivel, una biblioteca de pase de mensajes, una ambiente de comunicación de alta interoperabilidad y los servicios del metasistema.

Los **toolkits** para aplicaciones utilizan los servicios de la rejilla para proveer capacidades de alto nivel. Por ejemplo, el *toolkit* llamado "Remote Computation" fue creado por el equipo de desarrollo de Globus para ejecutar tareas distribuidas.

La última capa es la capa de **las aplicaciones**. En esta capa los usuarios desarrollan sus aplicaciones con la ayuda de los toolkits de la capa anterior. En la actualidad se han desarrollado aplicaciones de cosmología, ingeniería química, etc.

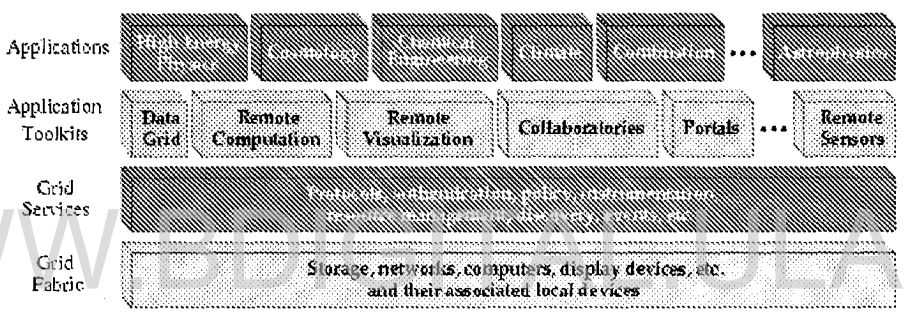

Figura 1.6: Arquitectura de Globus (Tomado de [18])

1.3.5. Condor

Condor [7] es un metasistema creado por la Universidad de Winsconsin en Mádison [2]. La arquitectura de Condor se muestra en la figura 1.7. Condor está compuesto por máquinas que someten tareas, un manejador central y las máquinas de ejecución. En lugar de ejecutar tareas que utilizan en forma intensiva la CPU de su estación de trabajo, un usuario somete sus tareas a Condor. Condor se encarga de encontrar una máquina disponible en la red para empezar a ejecutar el trabajo. Cuando Condor detecta que la máquina que ejecuta una tarea asignada ya no está a su disposición (por ejemplo porque se ha detectado actividad en el teclado de la

[2] http://www.cs.wisc.edu/condor

máquina de ejecución, o porque la maquina se encuentra muy cargada, o porque un usuario se ha conectado a la maquina de ejecución desde una máquina remota), Condor guarda el estado del trabajo (*checkpoint*) y lo transfiere a otra máquina de la red que esté inactiva. Condor sigue la ejecución del trabajo en la nueva máquina a partir de la última instrucción ejecutada. Si ninguna máquina está disponible en la red, entonces la tarea es almacenada en una cola en el manejador central hasta que una máquina este disponible. Al terminar la ejecución de una tarea, los resultados son transferidos al usuario por E-mail. El mecanismo de *checkpoint* de Condor consiste en hacer copias periódicas del estado de una tarea, aún cuando esta no sea transferida a otra máquina, con el objetivo de empezar dicha tarea desde su último estado guardado en caso de que la máquina deje de estar disponible para Condor, bien sea por un fallo o porque ésta pase a estar ocupada.

Condor implementa un interesante sistema de clasificados, parecido a los clasificados encontrados en los periódicos. Las máquinas de ejecución avisan al manejador central el tipo de aplicación que desean ejecutar, y especifican la cantidad de memoria máxima que pueden conceder al proceso que van a ejecutar. Las máquinas que someten tareas especifican las condiciones mínimas que deben cumplir las máquinas que ejecutarán sus tareas. Normalmente, un usuario Condor desarrolla su aplicación en su computador local, con el lenguaje deseado y lo copila localmente. El usuario puede ejecutar su programa en una máquina de ejecución con una arquitectura diferente a la de su máquina local. En este caso, utiliza un crosscompiler para compilar su aplicación para la arquitectura deseada. Luego, el usuario lanza la ejecución de la aplicación especificando la arquitectura de la máquina de ejecución. El manejador central empezará la ejecución de la tarea en una máquina de ejecución con la arquitectura especificada. En caso que ninguna máquina con la arquitectura deseada este disponible, la tarea queda en una cola del manejador central hasta poder ser atendida.

El ambiente de comunicación de Condor para la implementación de los servicios está basado en RPC (Remote Procedure Calls). Condor no tiene una biblioteca de pase de mensajes por lo que no permite la ejecución de programas paralelos, aunque se han reportado unas combinaciones con bibliotecas de comunicación como PVM.

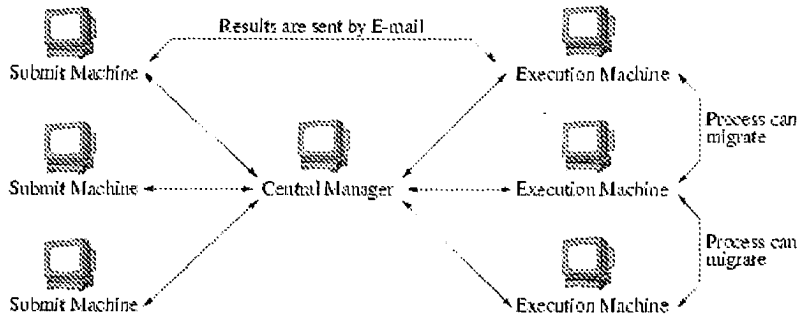

Figura 1.7: Arquitectura de Condor (Tomado de [15])

1.3.6. Popcorn

Popcorn [10] es un sistema de metacomputación desarrollado por la Universidad hebrea de jerusalen[3], Israel. Está hecho en Java y permite que cualquier computador que disponga de una implementación de la máquina virtual de Java participe en el metasistema. La funcionalidad básica de Popcorn es proveer al usuario de Internet de una máquina paralela virtual que funciona sobre redes con un pequeño ancho de banda, un tiempo de latencia alto y una fiabilidad baja. La arquitectura de Popcorn está representada en la figura 1.8. Popcorn está compuesto por tres entidades: (1) los compradores de tiempo de CPU, (2) los vendedores de tiempo de CPU y el market.

Los compradores de tiempo de CPU son programas de ejecución escritos en Java utilizando la API de Popcorn. Dicha API es parecida a los Java RMI (Remote Method Invocation) y los archivos de bytecode generados son llamados compuplets. Debido a que un vendedor de tiempo de CPU se puede salir de Popcorn sin avisar, los compuplets tienen mecanismos para ser lanzados de nuevo en otro vendedor de tiempo de CPU.

Los vendedores de tiempo de CPU están puestos a disposición de Popcorn para ejecutar los compuplets de los compradores de CPU. Para que un computador sea vendedor de tiempo de CPU, es suficiente que este visite la página Web de Popcorn para vendedores con un navegador especial de Java. Esta página tiene un applet que carga y ejecuta el código necesario para

[3]http://www.cs.huji.ac.il/labs/popcorn

convertir el computador en un vendedor de tiempo de CPU. Un vendedor puede salir de Popcorn cuando lo desee. Si dicho vendedor estaba ejecutando un compuplet, este último será lanzado de nuevo en otro vendedor.

El market en un lugar de encuentro entre compradores y vendedores de tiempo de CPU. Los compradores lo visitan con el objeto de comprar tiempo de CPU, y los computadores con el objetivo de vender tiempo de CPU. Se establece un sistema económico basado en el tiempo de CPU que es manejado por el market.

Un programa paralelo Popcorn está normalmente compuesto por un hilo de ejecución principal, el cual corre en la máquina local (comprador de tiempo CPU). Este hilo puede empezar un subconjunto de cómputos bien definidos en máquinas remotas (vendedores de tiempo CPU). Dicho conjunto de cómputos son un compuplet y encapsulan el código a ejecutar y los datos asociados al cómputo. Cada compuplet es ejecutado remotamente en una máquina vendedora de tiempo de CPU e informa al hilo principal al terminar. En Popcorn el hilo principal no sabe donde corren los compuplets que inicia, es por eso que el hilo principal debe trabajar en forma asíncrona, es decir que su ejecución no debe depender del orden de llegada de los resultados de los compuplets que inicia. Además, el hilo principal es informado de la pérdida de un compuplet cuando ésta ocurre, pérdida que puede ser debida a la caída de la comunicación o del vendedor de tiempo de CPU.

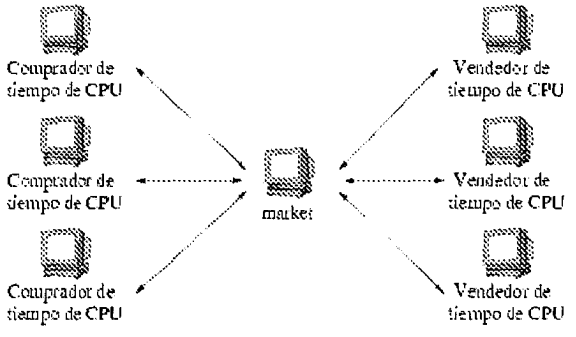

Figura 1.8: Arquitectura de Popcorn

1.3.7. Otros trabajos relacionados

Otros esfuerzos para explotar recursos distribuidos en sistemas de computación global son:

- Charlotte [6]: es un sistema que provee facilidades a los programadores de aplicaciones paralelas para escribir programas en Java y poder ejecutar programas paralelos desde un navegador Web. Los usuarios de Charlotte acceden a recursos computacionales distribuidos sin que sea necesario que estos sepan algo de redes de computadoras o de sistemas distribuidos.

- Javelin [13]: es una infraestructura de software para computación global basado en Java donde se administran recursos computacionales distribuidos. Provee un ambiente para la ejecución de aplicaciones paralelas, donde el cómputo paralelo se distribuye entre nodos de Internet.

- Atlas [5]: está diseñado para ejecutar programas paralelos multi-hilos haciendo uso de recursos computacionales distribuidos en la web. Es un sistema basado en Java cuyo objetivo es explotar los recursos ociosos de una red para presentar al usuario un gran sistema virtual distribuido.

- Globe [48]: es un sistema para manejo de recursos globales donde se implementan estrategias adaptativas de replicación de recursos web. Globe posee una lista de los servidores participantes en el sistema, éstos pueden ser manejados a nivel local y global. A nivel local el sistema maneja información relacionada a los recursos y servicios provistos por el servidor, mientras que a nivel global el sistema permite buscar, agregar y eliminar servidores de la lista del sistema.

Capítulo 2

Aspectos Teóricos

2.1. Generalidades sobre los Sistemas Operativos Web

En esta sección presentamos algunos fundamentos de los Sistemas Operativos en términos generales, para posteriormente tratar algunas ideas fundamentos de los Sistemas Operativos Web..

2.1.1. Fundamentos de los Sistemas Operativos

Un sistema operativo es un programa que actúa como intermediario entre el usuario y el hardware de un computador y su propósito es proporcionar un entorno en el cual el usuario pueda ejecutar programas[37]. El objetivo principal de un sistema operativo es lograr que el sistema de computación se use de manera cómoda, y que el hardware del computador se emplee de manera eficiente.

Las funciones de un sistema operativo son:

- proveer acceso compartido del(los) procesador(es) a los programas (o procesos).
- Maximizar el uso del tiempo del(los) procesador(es).
- Coordinar la ejecución de procesos que interactúan entre si.
- Administrar recursos computacionales (E/S, memoria, archivos, etc)
- Efectuar actividades de autenticación, control de acceso y protección.
- Mantener integridad del sistema.
- Recuperarse de fallas o errores.

- Proveer interfaz con usuario.

2.1.2. Estructura de un sistema operativo

Con el fin de tener una idea más extensa de cómo está estructurado un sistema operativo, veremos brevemente algunas estructuras de diseños de sistemas operativos que ya han sido probadas.

Estructura modular: Los sistemas operativos que presentan esta estructura son conocidos también como sistemas monolíticos. El sistema operativo se escribe como una colección de procedimientos, cada uno de los cuales puede llamar a los demás cada vez que así lo requiera. Cuando se usa esta técnica, cada procedimiento del sistema tiene una interfaz bien definida en términos de parámetros y resultados, y cada uno de ellos es libre de llamar a cualquier otro si este último proporciona cierto cálculo útil para el primero. Los servicios (llamadas al sistema) que proporciona el sistema operativo se solicitan colocando los parámetros en lugares bien definidos, como en los registros o en la pila, para después ejecutar una instrucción especial (trampa) de "llamada al núcleo" o"llamada al supervisor". Esta instrucción cambia la máquina del modo usuario al modo núcleo y transfiere el control al sistema operativo. El sistema operativo examina entonces los parámetros de la llamada, para determinar cual de ellas se desea realizar. A continuación, el sistema operativo analiza una tabla que contiene en la entrada k un apuntador al procedimiento que realiza la k-ésima llamada al sistema.

La estructura modular sugiere la siguiente organización básica del sistema operativo: 1.- un programa principal que llama al procedimiento del servicio solicitado. 2.- un conjunto de procedimientos de servicio que llevan a cabo las llamadas al sistema. 3.- un conjunto de procedimientos utilitarios que ayudan al procedimiento de servicio.

Estructura por micronúcleo: En los sistemas operativos que presentan esta estructura las funciones centrales son controladas por el núcleo, mientras que la interfaz del usuario es controlada por el entorno (*shell*).

Un ejemplo de sistema operativo micronúcleo es el DOS [32], en él, el componente más importante es un programa con el nombre "COMMAND.COM" Este programa tiene

dos partes: (1) El núcleo, que se mantiene en la memoria en todo momento, contiene el código máquina de bajo nivel para manejar la administración de hardware para otros programas que necesitan estos servicios; (2) el *shell*, el cual es el intérprete de comandos

En las estructuras por micronúcleo, las funciones de bajo nivel del sistema operativo y las funciones de interpretación de comandos están separadas, de tal forma que se puede mantener el núcleo del sistema operativo corriendo, y utilizar una interfaz de usuario diferente.

Estructura por capas: La estructura por capas consiste en organizar el sistema operativo en niveles jerárquicos o capas, donde cada capa es construida sobre la inmediata inferior. Existe una interfaz bien definida entre las capas. La capa más baja está más cerca del hardware y la capa más alta esta cerca del usuario.

Un ejemplo de sistema operativo por capas es el sistema MULTICS [49], el cual consta de 4 capas o anillos: (1) el núcleo del sistema, (2) llamadas al sistema, (3) bibliotecas del sistema, y (4) programas de usuario. (ver figura 2.1)

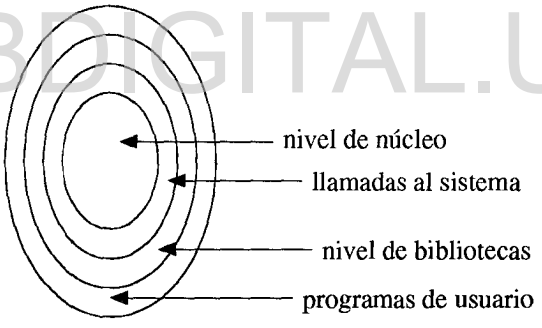

Figura 2.1: Estructura por capas del sistema operativo Multics (tomado de [49])

Una de las ventajas de los sistemas operativos estructurados por capas es que proveen modularidad lo que ha sido útil para el apoya del diseño del sistema operativo, ya que se puede comenzar a diseñar desde el nivel más bajo, para progresivamente codificar y probar cada nivel.

Estructura cliente - servidor Una tendencia de los sistemas operativos modernos es la de explotar la idea de mover el código a capas superiores y eliminar la mayor parte

posible del sistema operativo para mantener un núcleo mínimo. Con esta idea se busca implantar la mayoría de las funciones del sistema operativo en los procesos del usuario. Para solicitar un servicio, como la lectura de un bloque de cierto archivo, un proceso del usuario (denominado proceso cliente) envía la solicitud a un proceso servidor, que realiza el trabajo solicitado y devuelve una respuesta. En este modelo, el núcleo controla la comunicación entre los clientes y los servidores. Al separar el sistema operativo en partes, cada una de ellas controla una faceta del sistema, como el servicio a archivos, servicios a procesos, servicio a terminales o servicio a la memoria. Cada una de ellas es pequeña y controlable (ver figura 2.2).

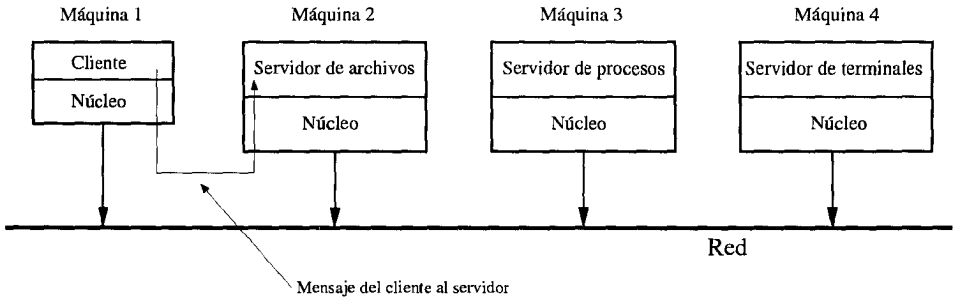

Figura 2.2: Estructura de un Sistema Operativo Cliente - Servidor

Una de las ventajas del modelo cliente-servidor es su capacidad de adaptación para su uso en los sistemas distribuidos. Si un cliente se comunica con un servidor mediante mensajes, el cliente no necesita saber si el mensaje se maneja en forma local, o si se envía por medio de una red a un servidor en una máquina remota. En lo que respecta al cliente, lo mismo ocurre en ambos casos: se envió una solicitud y se recibió una respuesta. Esta estructura de sistemas operativos marca una tendencia hoy en día hacia el desarrollo de los sistemas operativos distribuidos.

2.1.3. Sistemas operativos distribuidos

Un sistema operativo distribuido es aquel que aparece ante sus usuarios como un sistema tradicional de un solo procesador, aun cuando esté compuesto por varios procesadores [46]. En un sistema distribuido, los usuarios no deben ser conscientes del lugar donde su programa se ejecute o del lugar donde se encuentran sus archivos; eso debe ser manejado en forma

automática y eficaz por el sistema operativo [37].

Una implementación de un sistema operativo distribuido requiere más que simplemente agregar un poco de código a un sistema operativo mono-procesador, ya que los sistemas operativos distribuidos y centralizados difieren en aspectos críticos. Los sistemas distribuidos, por ejemplo, siempre permiten que las aplicaciones se ejecuten sobre varios procesadores al mismo tiempo, entonces se requieren de algoritmos de planificación de procesos más complejos con el fin de optimizar el paralelismo en el sistema.

En un sistema operativo distribuido los usuarios pueden acceder a recursos remotos de la misma manera en que lo hacen para los recursos locales. La migración de datos y procesos de un sitio a otra queda bajo el control del sistema operativo distribuido. Los sistemas operativos distribuidos permiten distribuir trabajos, tareas o procesos, entre un conjunto de procesadores. Puede ser que este conjunto de procesadores esté en un equipo o en diferentes, en este caso es transparente para el usuario. Los sistemas distribuidos deben de ser muy confiables, ya que si falla un componente del sistema, otro componente debe de ser capaz de reemplazarlo.

Algunas de las características de los Sistemas Operativos distribuidos son:

- *Transparencia:* proveer al usuario una máquina virtual.
- *Distribución de Servicios:* distribuir los servicios de manera eficiente.
- *Replicación de Servicios:* replicar los servicios para tolerar fallas.
- *Compartimiento de Recursos:* compartir recursos (discos, procesadores, memorias, etc.) de manera eficiente y confiable.
- *Coordinación:* coordinar todos estos servicios para evitar anomalías tales como los bloqueos o la muerte por inanición (*starvation*).
- *Servicios añadidos:* Denominación global, sistemas de archivos distribuidos, facilidades para distribución de cálculos (a través de la comunicación de procesos inter-nodos, llamadas a procedimientos remotos, etc.).

2.1.4. Sistemas Operativos Web

Los Sistemas Operativos Web (SOW) surgen como una instancia de los Sistemas Operativos Distribuidos. Un SOW es un sistema que administra de manera transparente recursos

computacionales distribuidos en la Web. En el diseño de un SOW, además de considerarse aspectos propios de los sistemas operativos distribuidos, se consideran otros aspectos particulares de los SOW, tales como:

- *Heterogéneidad:* Cómo administrar recursos en ambientes distribuidos heterogéneos donde no se conoce *a priori* las características de las arquitecturas o plataformas disponibles.
- *Volatilidad:* Cómo localizar los recursos disponibles, cuando éstos pueden cambiar de sitio, o pueden permanecer temporalmente fuera de línea, o incluso, se pueden agregar nuevos servicios en la Web.

A través del SOW, la Internet es manejada no sólo como un proveedor de información sino también como un proveedor de recursos computacionales, lo cual hace que el SOW pueda ofrecer las siguientes ventajas:

- *Alto Rendimiento:* Los recursos disponibles en la Web pueden ser aprovechados para el uso de aplicaciones de alto rendimiento (procesamiento paralelo, bases de datos distribuídas, etc.).
- *Capacidad de Crecimiento:* Los SOW son altamente escalables, por lo que el número de usuarios o nodos necesarios no debe ser visto como una limitación.
- *Soporte de Aplicaciones Inherentemente Distribuídas:* Por ejemplo, una empresa distribuida geográficamente puede aprovechar un SOW para dar soporte a sus sistemas distribuidos.
- *Carácter Abierto y Heterogéneo:* Facilidades para la interoperabilidad.

Por otro lado, el SOW posee las siguientes debilidades:

- Necesidad de un nuevo tipo de software más complejo.
- No hay todavía un acuerdo de cómo debe ser un SOW.
- La red es un elemento crítico. El SOW es susceptible a situaciones como los retardos, las pérdidas de mensajes, o la saturación de la red.
- La seguridad y la confidencialidad deben ser especialmente tratadas en los SOW.

2.2. Generalidades sobre el manejo de recursos en un SOW

Uno de los objetivos del SOW es administrar recursos computacionales distribuidos en la Web. Algunos sistemas globales para el manejo de recursos distribuidos han sido definidos en trabajos previos. De éstos trabajos se derivan tres enfoques de diseño:

- **Diseño basado en negociación de recursos.** El modelo de negociación propuesto en [47] coincide con el "modelo económico" propuesto en [3]. En este modelo hay tres clases de participantes: *Brokers*, Clientes y *Hosts*. Los *Brokers* coordinan la demanda y suministro de recursos computacionales, los *Hosts* ofrecen sus recursos como mercancía (registrandose en el *Broker*), y los clientes son los usuarios o los computadores que requieren de recursos adicionales. En este modelo, los recursos son manejados como mercancía negociable. Por ejemplo, un *Host* puede registrar (colocar en el mercado) 50 % de su tiempo de CPU de lunes a viernes, y 100 % de su tiempo de CPU los fines de semana. De acuerdo a los requerimientos del cliente, el *Broker* realiza las negociaciones pertinentes para llenar dichos requerimientos, para esto el *Broker* revisa en el mercado de recursos y determina cual es la mejor oferta, asignando así el recurso correspondiente al cliente (ver figura 2.3).

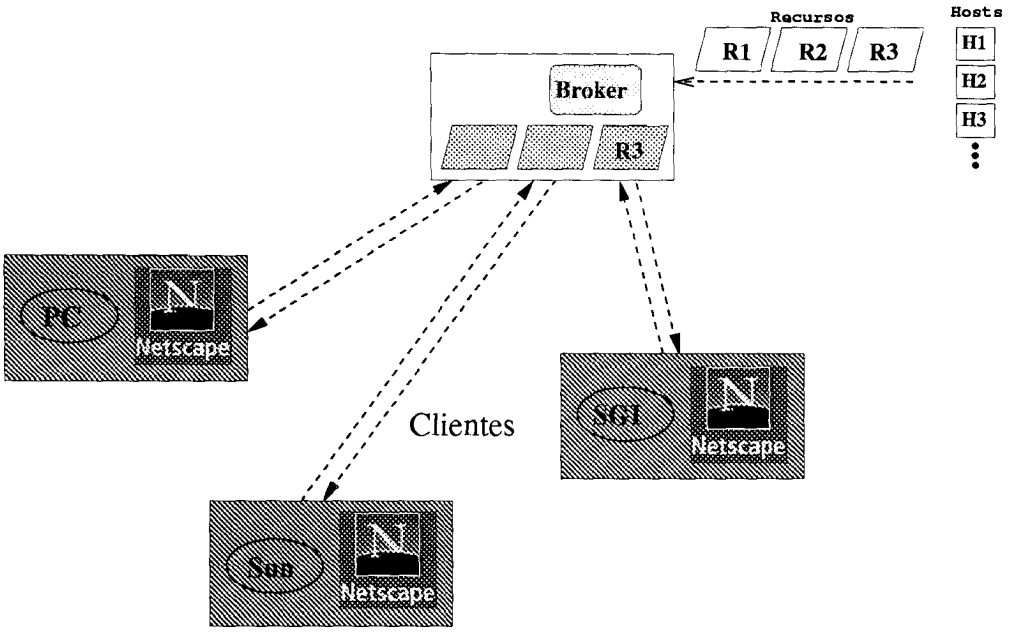

Figura 2.3: Arquitectura del modelo de negociación de recursos (Tomado de [3])

- **Diseño basado en conjuntos de recursos.** Este modelo se propone en [26] (ver figura 2.4). En esta arquitectura hay tres componentes fundamentales: el *conjunto de recursos*, los *servidores de recursos*, y la *interfaz de usuario*. El *conjunto de recursos* es una unidad central que sirve de espacio de coordinación para manejar recursos compartidos, funciona como un directorio público el cual contiene conjuntos y subconjuntos de recursos localizados en la red. Los conjuntos de recursos son implementados como objetos persistentes reusables, y pueden ser replicados y distribuidos en la red. El conjunto de recursos es responsable del seguimiento y la localización de recursos, éstos pueden ser dinámicamente agregados o removidos del conjunto de recursos, por ejemplo, si se detecta una falla permanente en un servidor de recursos, el conjunto de recursos remueve temporalmente el conjunto de recursos correspondientes. Otro de los componentes fundamentales de la arquitectura son los *servidores de recursos*, éstos servidores son responsables de hacer públicos a la Web los recursos locales y sus interfaces, registrandolos en los conjuntos de recursos. Finalmente, tenemos que el tercer componente fundamental de la arquitectura es la *interfaz de usuario*, su propósito es la interacción de los usuarios con la Web, y los *conjuntos de recursos*.

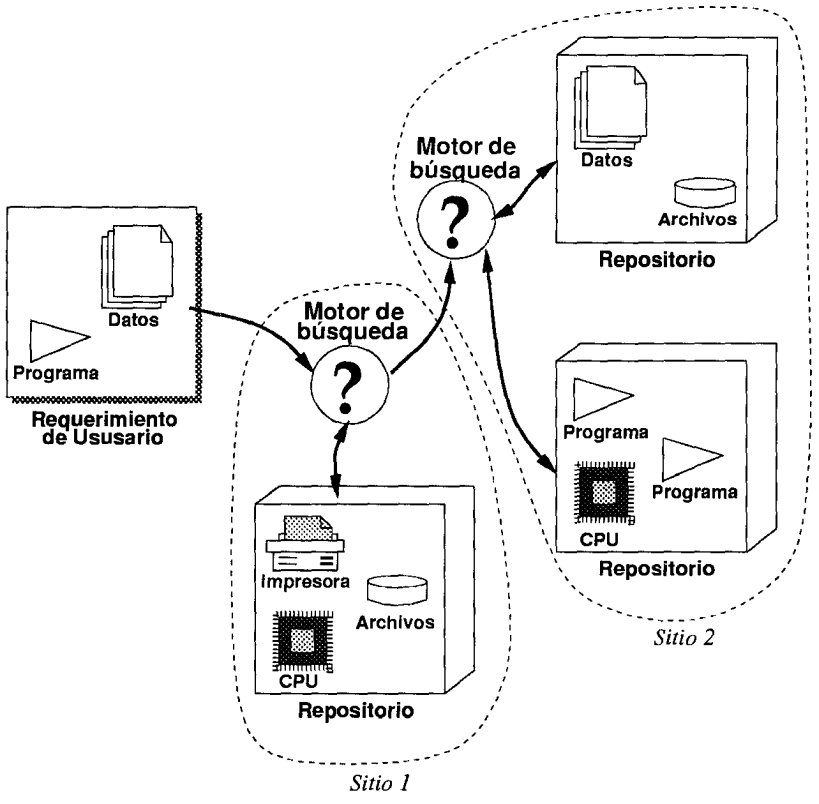

Figura 2.5: Un requerimiento de recurso manejado por dos motores de búsqueda

En el sistema para manejo de recursos del SOW que se propone en este trabajo, presentaremos un diseño orientado al uso de motores de búsqueda de recursos. En las siguientes secciones se describen algunas ideas fundamentales para el desarrollo de dicho diseño, como lo son el manejo de versiones de recursos y las búsquedas basadas en un modelo de inferencia.

2.3. Uso de versiones

La Web no solo es heterogénea a nivel conceptual (por la diversidad de servicios que ofrece), también lo es a nivel físico, es decir, a nivel del hardware y software disponible en diferentes sitios de la red. En la Web se pueden encontrar diversas tecnologías de red y diferentes máquinas, tales como computadores personales, estaciones de trabajos, supercomputadores, etc. Cada una de ellas con sus particulares configuraciones a nivel de sistemas operativos, protocolos y aplicaciones. No es fácil que un solo sistema operativo pueda administrar eficientemente la gran diversidad de recursos y servicios disponibles en la web, mas aún cuando

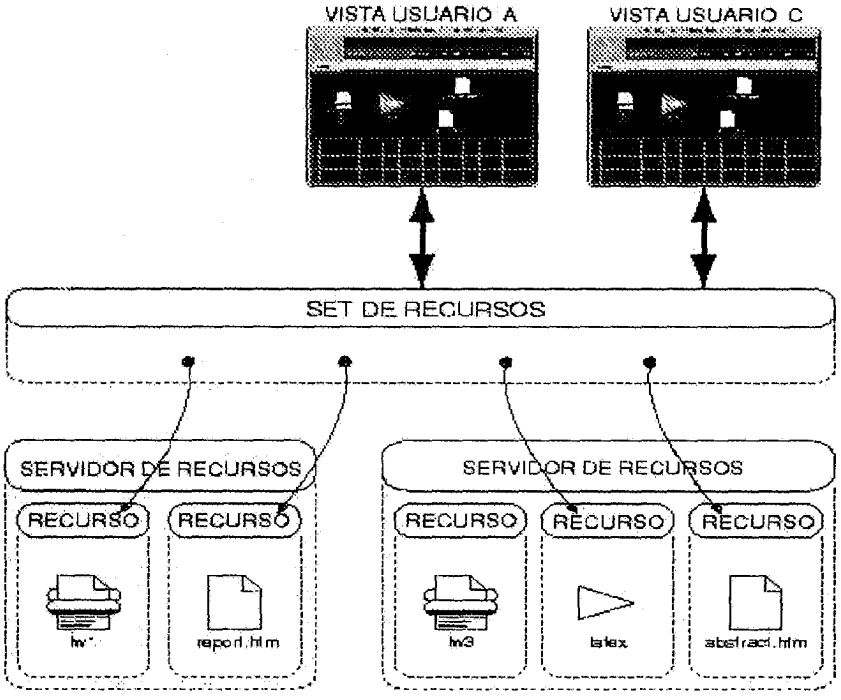

Figura 2.4: Arquitectura un modelo basado en conjuntos de recursos. Ejemplo de interacción entre un *conjunto de recursos*, dos *servidores de recursos* y dos *interfaces de usuarios* (Tomado de [26]).

- **Diseño basado en motores de búsqueda de recursos.** A diferencia de los diseños presentados anteriormente, en este caso las decisiones para la asignación de recursos computacionales no están centralizadas en una unidad específica. Un propuesta de este diseño se presenta en [27]. Los componentes fundamentales de una arquitectura basada en motores de búsqueda de recursos son los *repositorios* y los *motores de búsqueda*, ambos se encuentran distribuidos en la Web.

  Los *motores de búsqueda* son sistemas reactivos que responden a requerimientos provenientes de usuarios o de otros *motores de búsqueda*, y satisfacen dichos requerimientos haciendo uso de la información contenida en sus *repositorios*. Todo requerimiento de recurso pasa por los *motores de búsqueda*, los cuales recorren la red con el fin de conseguir el recurso solicitado. En este proceso, los *motores de búsqueda* son asistidos por sus *repositorios*, los cuales proveen la información necesaria para la localización de los recursos (ver figura 2.5).

sabemos que éstos no son fijos, debido a los constantes cambios que experimenta la web tanto a nivel conceptual como a nivel físico.

Tomando en cuenta lo anterior, se ha sugerido que un SOW debe ser diseñado no solo como un *sistema distribuido*, sino también como un *sistema versionado* [30]. Es decir, un sistema que tenga diferentes versiones de los servicios corriendo simultáneamente en diferentes nodos de la red. Para el diseño de un SOW, el manejo de versiones es un factor clave, a continuación se explican algunos aspectos importantes relacionados a esta técnica y su aplicación en la construcción de sistemas distribuidos versionados.

2.3.1. Control de versiones y configuración de software

Los tipos de sistemas cuyos componentes vienen en varias versiones, y para los cuales se pueden admitir múltiples configuraciones, se conocen como sistemas versionados. La idea del uso de versiones en el desarrollo de software no es nada nueva, históricamente los sistemas de software han evolucionado como producto de cambios que progresivamente se introducen en los mismos. Siempre es necesario corregir errores o implementar cambios de especificaciones para mejorar el software. Surgen entonces diferentes versiones de software para llenar las mismas o nuevas necesidades. En los grandes sistemas, sus componentes no siempre evolucionan al mismo ritmo, así, el control de versiones en estos grandes sistemas se torna complejo, y es por esto que se requiere de mecanismos que permitan controlar las diferentes versiones de los componentes. El adecuado control de versiones se logra a través del proceso de configuración del software.

A través del proceso de configuración del software se construye un sistema o subsistema a partir de sus componentes básicos. Si cada componente viene en una simple versión, el problema sería bastante sencillo; pero el caso más común es cuando cada componente tiene diferentes versiones. Cuando se configura el software, se requiere de versiones particulares de cada uno de sus componentes para construir un sistema dado. Es decir, antes de construir el sistema se deben seleccionar las versiones apropiadas de cada componente para luego integrarlas y obtener la configuración apropiada. En [29] se cita como un ejemplo de un sistema de configuración de software muy conocido a la utilidad *make* [17], su propósito es compilar e integrar los componentes de un programa. La utilidad *make* determina automáticamente

qué piezas de un programa necesitan ser recompiladas, y lanza las órdenes para recompilarlas. Es decir, permite reconfigurar automáticamente el software cuando se han realizado cambios a alguno de sus componentes. Así, cuando se introduce una nueva versión de un componente, el *make* puede realizar la configuración para integrar esta nueva versión al sistema y obtener la nueva configuración del software.

Sabemos ahora que los sistemas versionados admiten múltiples configuraciones, pero, ¿Cual será la configuración más adecuada?, ¿Como escoger la versión apropiada de cada componente del sistema?. Las respuestas a estas interrogantes no son difíciles de conseguir, ya que las versiones de software se implementan para satisfacer ciertas necesidades o requerimientos, y el fabricante del software maneja esta información de manera muy precisa. Cuando se dispone de esta información, es posible llevar una relación clara entre los requerimientos y la versión adecuada para dicho requerimiento. Entonces, en función de esta información se obtiene la configuración deseada. Volviendo al ejemplo del sistema de configuración *make*, éste realiza la configuración en función de cierta información extraída de un repositorio de datos, que no es más que un archivo de texto conocido como *makefile*. En ese archivo se le suministra al sistema *make*, a través de ciertos parámetros, toda la información inherente al manejo de las versiones de los componentes del software.

2.3.2. El modelo de inferencia en el manejo de versiones

Para obtener una configuración automática de software se usa un modelo computacional conocido como "modelo de inferencia". Éste es un proceso de manejo de demandas, el cual fue empleado originalmente en el área de Sistemas Expertos [23]. Hoy en dia su uso se ha extendido a otras áreas y ha sido aplicado para la solución de problemas tales como, el manejo de plataformas computacionales heterogéneas y el manejo de versiones [38].

La técnica de "inferencia" se basa en un proceso computacional muy natural: un usuario realiza una consulta de cierta información, la cual puede ser devuelta de inmediato, o posiblemente, esta consulta inicial genere nuevas demandas para poder obtener la información solicitada. Este es, por ejemplo, el proceso que se lleva a cabo cuando se consulta una información al *World Wide Web*. Un elemento clave en ésta técnica son los "repositorios de inferencia", los cuales proveen información útil para el manejo de las demandas. A través de

los repositorios se pueden tener *caches* de la información previamente consultada, con lo que se evita repetir ciertas consultas, y por lo tanto, generar una respuesta más rápida [27].

El proceso de inferencia para el manejo de versiones se inicia con la demanda de algún programa. Posteriormente se realiza la configuración de la versión del programa o software requerido, para esto se necesitan las versiones apropiadas de los subsistemas o componentes del software. Para cada subsistema se requieren las versiones de sus componentes, y así sucesivamente, hasta que se obtengan las versiones de los componentes atómicos. Una vez que todos los componentes y sus respectivas versiones han sido identificados, se procede a la compilación o construcción de la versión de software requerida. En este proceso, los motores de búsqueda reciben los requerimientos de versiones, buscan en sus repositorios y devuelven la versión requerida si la poseen, en caso contrario, realizan una nueva consulta a otros motores de búsqueda.

A través de este modelo computacional, el SMR del SOW actúa como un ente tomador de decisiones. Así tenemos que como soporte a las tomas de decisiones, el uso de técnicas inteligentes para el manejo de recursos resulta oportuno. Así, el SMR del SOW puede ser concebido como un Sistema Inteligente Distribuido, el cual puede ser desarrollado bajo un enfoque de Sistemas Multiagentes. En la siguiente sección introducimos algunas ideas fundamentales de los Sistemas Multiagentes.

2.4. Fundamentos de los Sistemas Multiagentes

2.4.1. Inteligencia Artificial Distribuida (IAD)

La Inteligencia Artificial Distribuida (IAD) [9, 12, 34] se basa en la interacción de diversas entidades. La investigación en el campo de la Inteligencia Artificial ha ayudado a desarrollar software que simula las capacidades de la inteligencia del ser humano, tales como razonamiento, comunicación en lenguaje natural, aprendizaje, etc. Con tales programas, las computadoras han pasado progresivamente a ser una especie de asistente para las personas. La Inteligencia Artificial Distribuida (IAD) es un subcampo de la Inteligencia Artificial que investiga modelos de conocimiento, así como técnicas de comunicación y razonamiento que se basan en la integración de agentes computacionales para participar en la solución de prob-

lemas. De esta forma, la investigación en IAD se orienta al entendimiento y modelado de conocimiento y acción en tareas que requieren de la colaboración. De aquí existen dos áreas de investigación [22]: Solución de Problemas Distribuidos, y Sistemas Multiagentes. La investigación en el área de Soluciones Problemas Distribuidos (SPD) considera la forma en que se puede dividir la tarea de solucionar un problema dado, a través de un número de módulos (o nodos) que cooperan entre sí. En un sistema de SPD puro, todas las estrategias de interacción (cooperación y coordinación) se incorporan como parte integral del sistema. La investigación en Sistemas Multiagentes (SMA) se orienta al estudio del comportamiento de un conjunto de agentes autónomos que tratan de dar solución a un problema dado. En un sistema SMA puro, se facilita la cooperación y coordinación entre los nodos del sistema mediante un plan que especifica las acciones e interacciones de los agentes. La Solución de Problemas Distribuidos y los Sistemas Multiagentes coinciden en el interés del estudio de entidades distribuidas. Sin embargo, en la Solución de Problemas Distribuidos inicialmente se define una tarea global y el problema consiste en diseñar las entidades distribuidas que sean capaces de efectuar esa tarea global, para eso es necesario estudiar la distribución y los puntos de colaboración. En los Sistemas Multiagentes, se definen inicialmente entidades autónomas y se estudia la forma en que estas entidades son capaces de realizar tareas en conjunto. La característica principal consiste en estudiar la estructura de las entidades (o agentes) autónomas y cómo pueden cooperar entre sí.

2.4.2. Sistemas Multiagentes (SMA)

Un Sistema Multiagente [45] puede definirse como una red de "solucionadores de problemas" que trabajan juntos para dar solución a problemas que están más allá de sus capacidades individuales. Estos solucionadores de problemas, a los cuales se les llama "agentes", son autónomos y pueden ser homogéneos o heterogéneos en su naturaleza.

2.4.2.1. Ventajas de los SMA

Un Sistema Multiagente tiene grandes ventajas con respecto a un sistema centralizado y monolítico:

- Solución de problemas con mayor rapidez, debido al aprovechamiento del procesamiento

paralelo.

- Comunicación mínima, pues se transmite solamente soluciones parciales de alto nivel a otros agentes en lugar de tener que enviar datos básicos a un sistema central.
- Mayor flexibilidad, pues se tienen agentes con diferentes habilidades que en forma dinámica cooperan entre sí para resolver problemas.
- Mayor confiabilidad, pues otros agentes pueden tomar las responsabilidades de los agentes que llegasen a fallar en su operación.

2.4.2.2. Problemas en el diseño de SMA

Hay seis problemas inherentes al diseño e implementación de Sistemas Multiagentes:

1. Cómo formular, describir, descomponer y asignar problemas y sintetizar resultados entre grupos de agentes.
2. Cómo lograr la comunicación e interacción entre agentes. Cuáles lenguajes y protocolos de comunicación utilizar para Sistemas Multiagentes. Cuándo y qué comunicar.
3. Cómo asegurar que los agentes actúen coherentemente en la toma de decisiones o en la ejecución de acciones, de forma tal que se eviten interacciones dañinas debidas a decisiones particulares que pudiesen afectar decisiones globales.
4. Cómo permitir que agentes individuales representen y razonen sobre los planes, acciones y conocimientos de otros agentes con el propósito de coordinarse con ellos. Cómo razonar acerca del estado de sus procesos coordinados (p.ej. iniciación, ejecución, y terminación).
5. Cómo reconocer y reconciliar perspectivas opuestas e intenciones en conflicto entre agentes tratando de coordinar sus acciones.
6. Cómo diseñar y desarrollar sistemas distribuidos prácticos. Cómo diseñar plataformas tecnológicas y metodologías de desarrollo para SMA.

2.4.2.3. Características de los Agentes

Debido a que la definición de agente ha resultado ser tan controvertida como la definición de inteligencia artificial, se ha optado por la definición de un conjunto de propiedades que

caracterizan a los agentes, aunque un agente no tiene que poseer todas estas propiedades [8]. Las características más importantes son:

Autonomía: Capacidad de operar sin intervención directa de los humanos o de otros agentes, con un cierto tipo de control sobre sus acciones. La autonomía es una de las características más importantes, esta le permitirá definir su conducta basado en su propia experiencia.

Sociabilidad: Los agentes son capaces de interactuar con otros agentes, a través de un lenguaje de comunicación entre agentes.

Reactividad: Las percepciones captadas de su ambiente producen una acción especifica.

Proactividad: Los agentes no sólo son entidades que reaccionan a un estímulo, sino que tienen capacidad de exhibir un comportamiento particular dependiendo de los objetivos planteados.

Continuidad: Los agentes están constantemente ejecutando procesos (captando percepciones y ejecutando acciones).

Benevolencia: Capacidad de satisfacer solicitudes. Se supone que los agentes no tienen metas conflictivas, y por lo tanto harán siempre lo que se les pide que hagan.

Aprendizaje: Los agentes, para ser inteligentes, requieren tener la propiedad de poder aprender del ambiente que les rodea.

Movilidad: Algunos agentes de software pueden tener la habilidad de viajar en una red de computadoras como por ejemplo la Internet.

Racionalidad: El agente actuará de manera tal de satisfacer sus objetivos.

Colaboración: Al interactuar de manera constante con otros agentes, un agente puede solicitar la colaboración de otros con la finalidad de ejecutar acciones eficaces y eficientes.

2.4.2.4. Agentes Inteligentes

Se consideran a los agentes inteligentes [35] como una pieza de software que ejecuta una tarea dada utilizando información recolectada del ambiente, para actuar de manera apropiada

hasta completar la tarea de manera exitosa. El software debe ser capaz de autoajustarse basándose en los cambios que ocurren en su ambiente.

Los agentes inteligentes son racionales, es decir, hacen lo correcto. De esta afirmación se desprende la pregunta, ¿Qué es lo correcto?. Lo correcto es lo que le permite al agente obtener el mejor desempeño. Para evaluar el desempeño es necesario decidir cómo y cuándo medir dicho desempeño.

El cómo se refiere al criterio que sirve para definir qué tan exitoso ha sido un agente en la consecución de los objetivos para el cual fue programado; ésta medida se establece como una norma por parte del diseñador, y la comparación contra la misma permite determinar la satisfacción de desempeño del agente. El cuándo se refiere al tiempo empleado en la realización de la tarea que se considere aceptable, dependiendo del ambiente de acción.

La principal característica de los Agentes Inteligentes es el conocimiento que estos poseen, aunado a la forma como lo utilizan para alcanzar las metas para la cual fueron diseñados.

Un agente inteligente puede ser descrito por los siguientes atributos:

- Sus tareas: el conjunto de funciones y actividades de los agentes, las cuales le permiten cumplir sus objetivos.
- Sus comunicaciones: define su forma de interacción con el ambiente y con otros agentes.
- Su conocimientos: las habilidades e información que poseen los agentes para realizar sus tareas.

2.5. Metodología para el desarrollo de SMA

En este trabajo utilizamos una metodología para el desarrollo de Sistemas Multiagente [2], la cual es una extensión de la metodología MAS-*Common*KADS [21]. A continuación se presentan algunas generalidades de ésta metodología.

2.5.1. La propuesta de MAS-*Common*KADS extendido

La identificación de una metodología de ingeniería de software normalmente no parte de cero, si no que es un proceso de refinamiento, añadiendo los nuevos aspectos y perspectivas de los sistemas y lenguajes e integrando los ingredientes exitosos de metodologías previas,

ésta será la aproximación seguida aquí. Tal y como se presentó en la sección 2.4 el concepto de agente se puede ver como un elemento integrador de técnicas orientadas a objetos y de los sistemas basados en conocimiento. Por una parte, los agentes extienden el paradigma orientado a objetos proporcionando un nivel de abstracción mayor que los objetos, el agente, a quien pueden atribuirsele determinadas propiedades y razonar con estas propiedades. Con esta perspectiva integradora, revisamos las principales metodologías orientadas a objeto y orientadas a agente, y decidimos tomar como marco de referencia de la metodología MAS-*Common*KADS extendido ésta es una extensión de la metodología MAS-*Common*KADS, la cual añade aspectos que son relevantes para los sistemas multiagentes y que no fueron cubiertos en la propuesta de MAS-*Common*KADS.

2.5.2. Descripción de Metodología MAS-*Common*KADS extendido

En este apartado presentaremos una visión general de la metodología propuesta en [2]. Describiremos brevemente sus modelos y el ciclo de desarrollo propuesto.

2.5.2.1. Modelos de MAS-*Common*KADS extendido

La metodología MAS-*Common*KADS extendido propone los siguientes modelos para el desarrollo de sistemas multiagentes (ver figura 2.6).

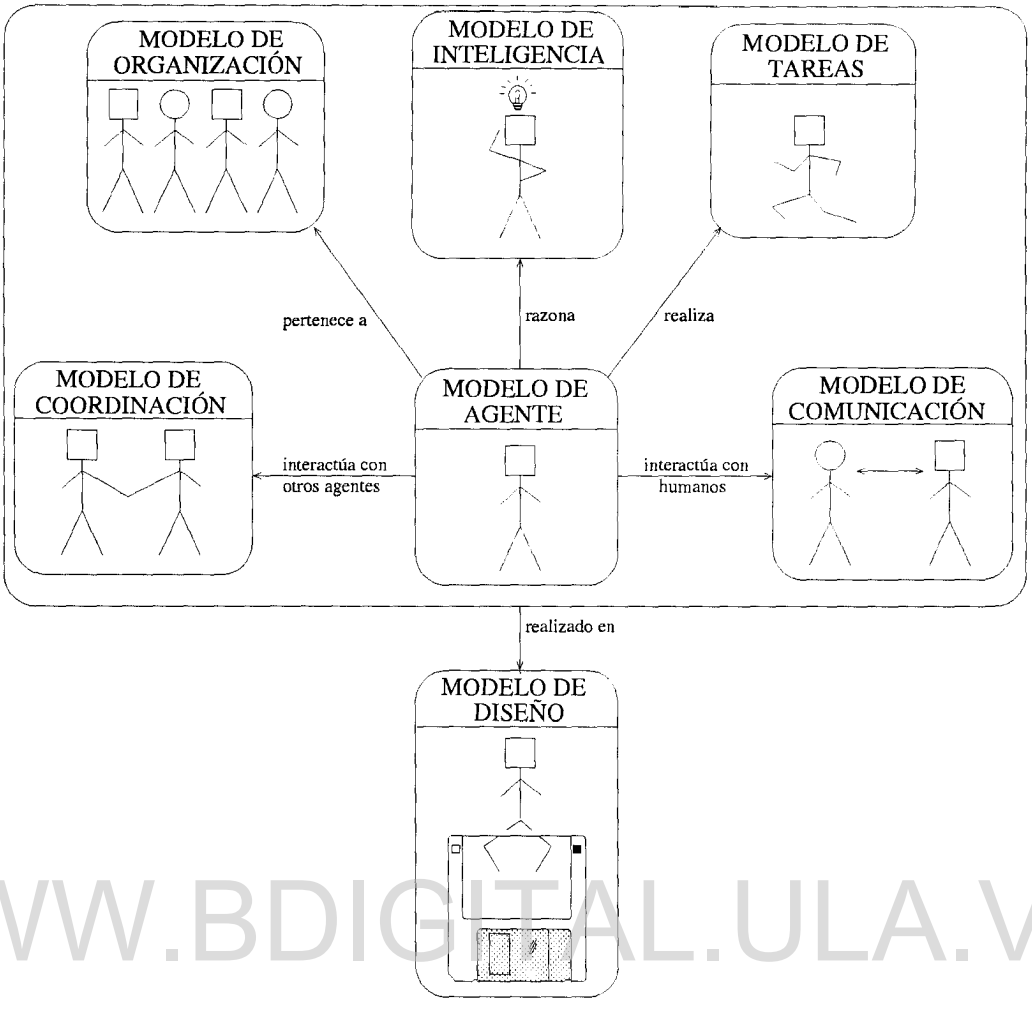

Figura 2.6: Los Modelos de MAS-*Common*KADS extendido

Modelo de Agente (MA): es el mismo modelo presentado en [21], este modelo especifica las características de un agente: sus capacidades de racionamiento, habilidades, servicios, sensores, efectores, grupos de agentes a los que pertenece, entre otras cosas. Un agente puede ser un agente humano, de software, o cualquier entidad capaz de emplear un lenguaje de comunicación de agentes.

Modelo de Organización (MO): es el mismo modelo presentado en [21], este modelo sirve como una herramienta para analizar la organización humana en que la el sistema multiagentes va ser introducido y para describir la organización de los agentes de software y su relación con el entorno.

Modelos de Tareas (MT): describe las tareas que los agentes pueden realizar: los objetivos de cada tarea, su descomposición, los ingredientes y los métodos de resolución de problemas para resolver cada objetivo. El modelo de tareas se extiende con la incorporación de una plantilla de especificaciones: el **método** asociado a la tarea (ver figura 2.7). A través de él se describen los métodos o algoritmos empleados para la realización de una tarea específica.

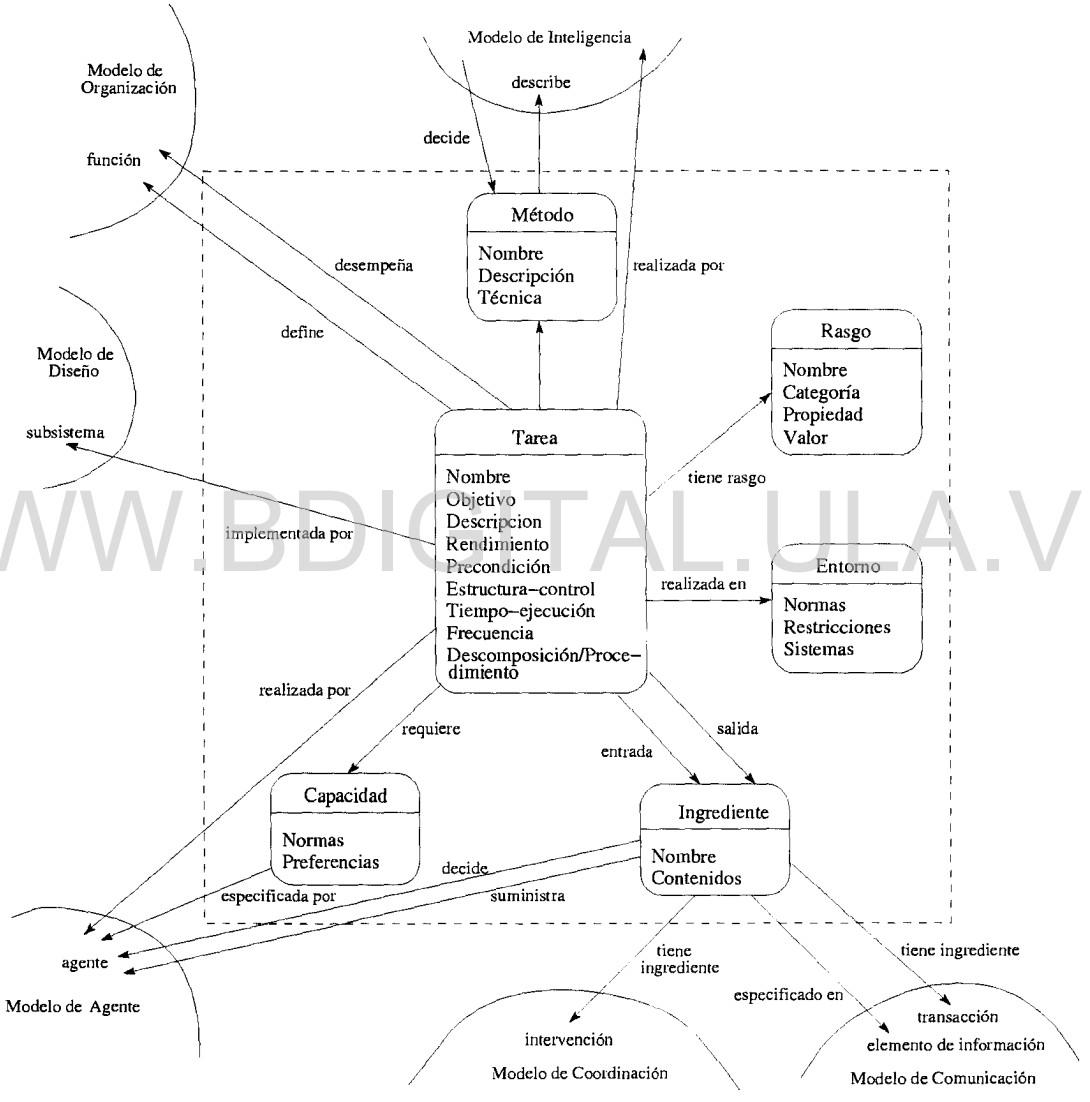

Figura 2.7: Modelo de tareas de MAS-*Common*KADS extendido

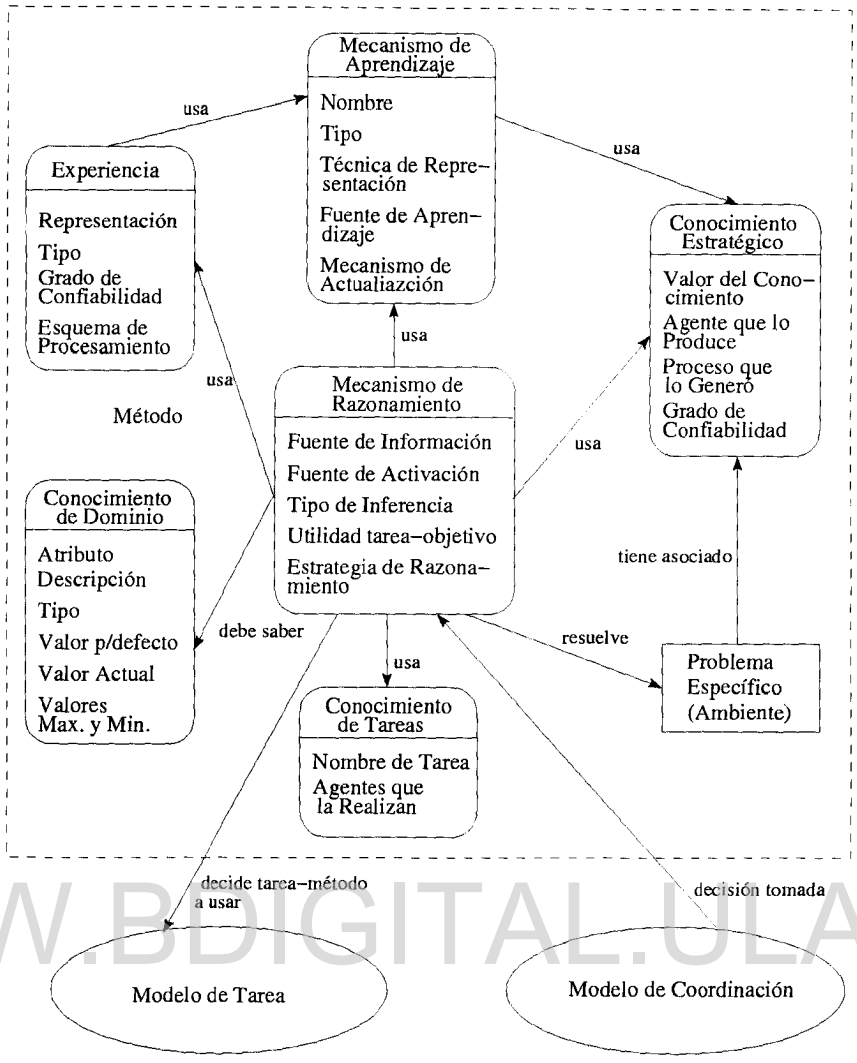

Figura 2.8: Modelo de inteligencia de MAS-*Common*KADS extendido

Modelo de Inteligencia(MI): describe el conocimiento necesitado por los agentes para alcanzar sus objetivos. En este modelo se incorporan especificaciones que no fueron consideradas en MAS-*Common*KADS, tale como: **mecanismo de razonamiento** para definir aspectos como la fuente información manejada, el tipo de inferencia, y la estrategia de razonamiento usada, entre otros; **mecanismo de aprendizaje** para definir la fuente de aprendizaje, y las técnicas de actualización y representación del aprendizaje; **conocimiento estratégico** para definir el agente que produce un conocimiento fundamental, el valor de dicho conocimiento, y el grado de confiabilidad, entre otros; **conocimiento de tareas** que contiene información de las tareas fundamentales y que

agentes la realizan; y la **experiencia** para definir la forma de representar la experiencia, el esquema de procesamiento, el tipo de experiencia y el grado de confiabilidad de la experiencia generada (ver figura 2.8).

Modelo de Comunicación (MC): describe las interacciones entre los agente de software. El *modelo de comunicación* de MAS-*Common*KADS extendido es el resultado de una fusión de los modelos de comunicación y coordinación de MAS-*Common*KADS.

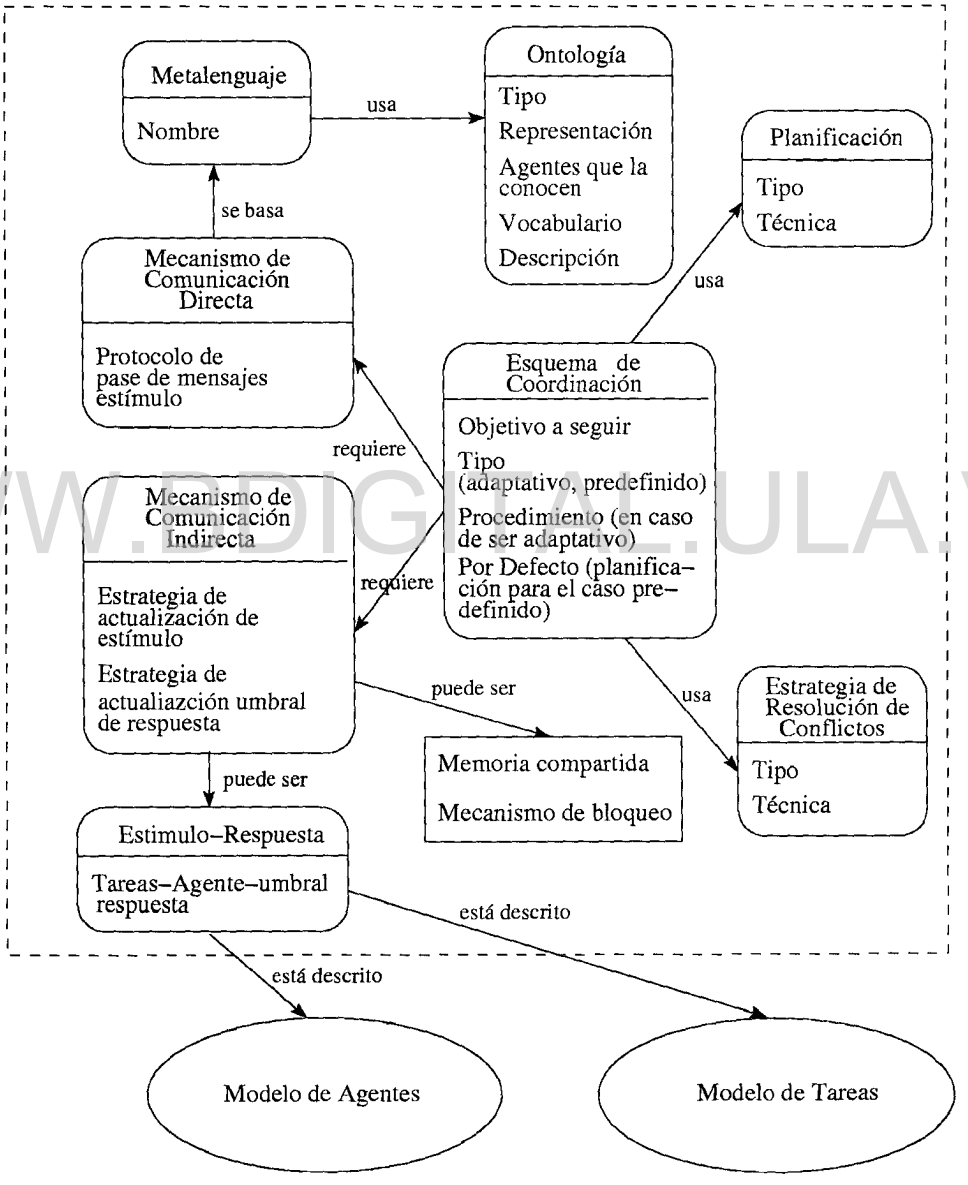

Figura 2.9: Modelo de coordinación de MAS-*Common*KADS extendido

Modelo de Coordinación (MCo): a través de este modelo se representa la estructura de comunicación del sistema, además de los protocolos y lenguajes asociados a las comunicaciones. En este modelo se incluyen especificaciones que no fueron presentadas en MAS-*Common*KADS, tales como: **esquema de coordinación** para definir el tipo de coordinación usada (adaptativa o predefinida), el objetivo a seguir, el procedimiento usado en la coordinación adaptativa, y la planificación por defecto usada en la coordinación predefinida; **planificaión** para definir el tipo y técnica de planificación usada; **mecanismo de comunicación directa** para definir protocolos y estímulos usados para la comunicación directa; **mecanismo de comunicación indirecta** para definir la estrategia de actualización de estímulo y de umbral de respuesta en las comunicaciones indirectas; **metalenguaje** para definir el metalenguaje usado para coordinar los agentes; y **ontología** para definir el vocabulario manejado. (ver figura 2.9).

Modelo de Diseño (MD): es el mismo modelo presentado en [21]. Mientras que los otros cinco modelos tratan sobre la especificación del sistema multiagentes, este modelo se utiliza para describir la arquitectura y el diseño del sistema multiagentes como paso previo a su implementación.

2.5.2.2. Fases de desarrollo de MAS-*Common*KADS extendido

El modelo de ciclo de vida para el desarrollo de sistemas multiagentes usando MAS-*Common*KADS extendido sigue las siguientes fases:

- **Conceptuación:** tarea de extracción o adquisición de conocimiento para obtener una primera descripción del problema y la determinación de los casos de uso que pueden ayudar a entender los requisitos informales y probar el sistema. Durante la fase de conceptuación se concibe el problema que se va resolver y se elabora un primer esbozo del sistema que puede resolverlo. En esta metodología se propone el desarrollo de la fase de conceptuación a través del uso de uno de los enfoques mas extendidos de las metodologías orientadas a objetos: el análisis centrado en el usuario.

- **Análisis:** determinación de los requisitos de nuestro sistema partiendo del enunciado del problema. Durante esta fase se desarrollan los siguientes modelos: organización, tareas, agentes, comunicación, coordinación e inteligencia. Se describe la estructura de cada

modelo a través de plantillas textuales y otras herramientas de modelado como las que provee UML.

- **Diseño:** determinación de cómo los requisitos de la fase de análisis pueden ser logrados mediante el desarrollo del modelo de diseño. Se determinan las arquitecturas tanto de la red multiagentes como de cada agente.

- **Codificación y prueba:** un desarrollador de software se encarga de escribir los programas y agentes diseñados. Durante esta fase, el sistema se emplea de manera experimental para asegurarse que el software no tenga fallas, es decir, que funciona de acuerdo con las especificaciones y en la forma en que los usuarios esperan que lo haga.

- **Integración:** todos los componentes del sistema son probados. Las partes son ensambladas en conjuntos progresivamente más grandes.

- **Operación y mantenimiento:** esta etapa comprende la puesta en funcionamiento del sistema, y la detección y corrección de errores no observados en las etapas anteriores. También pueden producirse cambios en el sistema para adaptarlo a la evolución del entorno o a nuevos requisitos.

Capítulo 3

Propuesta de un Sistema Manejador de Recursos

3.1. Descripción del SOW propuesto

3.1.1. Características del SOW

El SOW propuesto posee la siguientes características:

- Distribuido y versionado: Diferentes versiones de los servicios que el SOW utiliza en su proceso de configuración dinámica están corriendo simultáneamente sobre la red.
- Dinámico: La web es una entidad que está evolucionando permanentemente, por lo tanto, el SOW debe adaptarse a los cambios imprevistos. Es por ello que nuestro SOW tiene incorporado cierto dinamismo en los subsistemas que lo integran. Por ejemplo, a través de la configuración dinámica de los servicios que ofrece; a través de la actualización dinámica de la información sobre los recursos locales y remotos disponibles en cada nodo; a través de la agrupación dinámica de los nodos existentes de acuerdo con ciertas características, y a través de la migración, replicación y seguimiento de objetos web.
- Abierto: Nuestro SOW se caracteriza como un sistema abierto desde dos puntos de vista. En primer lugar, aceptará que diversas tecnologías sean usadas en todos los niveles de la red (heterogeneidad). En segundo lugar, permitirá que cualquier nodo de Internet pueda ser incorporado al sistema.
- Inteligente: Cada uno de los subsistemas del SOW podrá tener algún nivel de inteligencia

para el desarrollo de algunas de sus funciones, con el fin de optimizar su funcionamiento.

Nuestro SOW es un sistema operativo versionado e inteligente que se autoconfigura dinámicamente para permitir un acceso fácil y transparente a los recursos distribuidos sobre la Internet. El SOW utiliza un *motor de búsqueda* como un sistema reactivo que maneja las demandas, a través del cual se administran los recursos del ambiente computacional heterogéneo de la web.

Cada vez que un requerimiento llega a un nodo del SOW, los repositorios participan activamente en el procesamiento del requerimiento. En el SOW propuesto intervienen dos tipos de repositorios por cada nodo: los **repositorios de recursos locales** como dispositivos de almacenamiento que guardan y actualizan la información sobre los recursos disponibles localmente y, los **repositorios de recursos remotos** como dispositivos de almacenamiento de la información remota a la que se accede con frecuencia. Los repositorios cuentan con mecanismos para mantener actualizada y coherente la información almacenada. En general, los repositorios locales son los primeros repositorios consultados por los motores deductivos, ya que la prioridad de éstos es ejecutar los servicios solicitados localmente en la medida de lo posible para ofrecer tiempos de respuesta aceptables.

Por otro lado, debido al gran dinamismo presente en la web y a la gran cantidad de nodos que puedan estar conectados al SOW, se crean **comunidades** que son conjunto de nodos pertenecientes al SOW que exhiban afinidades funcionales y conductuales, los cuales son capaces de asociarse y disociarse dinámicamente. La utilización de comunidades permitirá optimizar la búsqueda de algún servicio, ya que en lugar de buscar nodo por nodo se podría buscar comunidad por comunidad.

Por último, nuestro SOW controla la gestión de objetos web cuya ubicación debe estar cerca de los sitios de mayor demanda en la Internet para ofrecer mejores tiempos de respuesta, a través de mecanismos que soportan **objetos móviles** capaces de decidir autónoma e inteligentemente a donde moverse o migrar, y de mecanismos de replicación de objetos que determinen de manera autónoma cuando replicar o eliminar un objeto en una entidad distinta a donde se originó el objeto.

3.1.2. Integración del SOW en un ambiente distribuido

Nuestro SOW debería estar al tope de una plataforma estándar distribuida (Middleware) para aprovechar los servicios provistos por dicha plataforma, tales como los mecanismos de nombramiento, seguridad, entre otros. (ver figura 3.1).

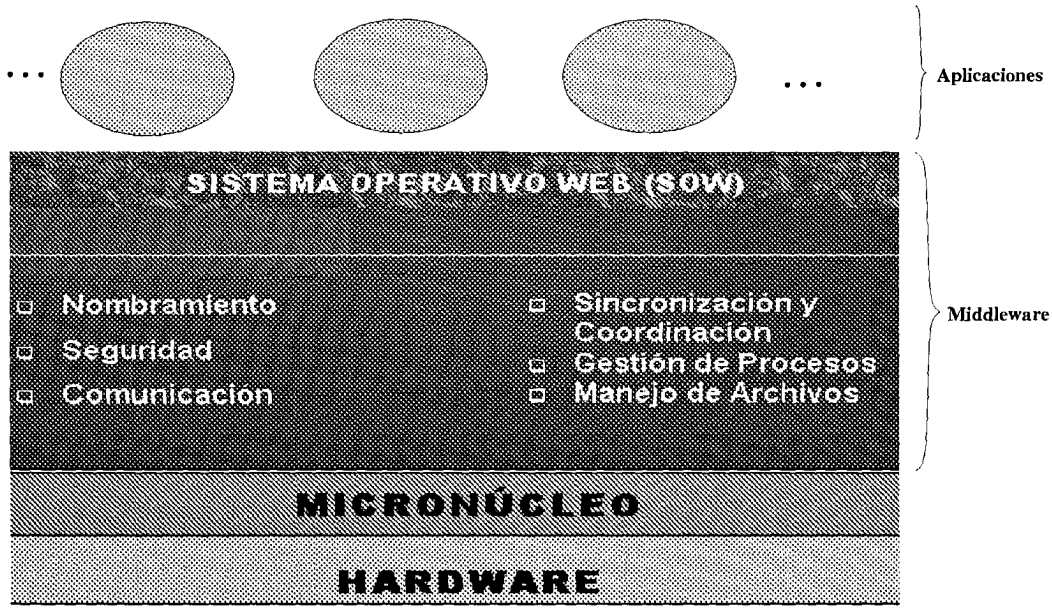

Figura 3.1: Esquema de integracón del SOW propuesto en un ambiente distribuido

Particularmente, entre los servicios de entorno distribuido que deberán ser provistos a nuestro SOW por la plataforma estándar se encuentran: un **Servicio de Nombres** para la identificación y localización de recursos en el entorno distribuido por nombramiento; un **Servicio de Directorio** para la identificación y localización de recursos en el entorno distribuido a través de búsquedas por atributo; un **Servicio de Seguridad** que proporcione *confidencialidad* (protección contra usuarios no autorizados), *integridad* (protección contra alteraciones o corrupción de la información), y *disponibilidad* (protección contra interferencia en los medios para acceder los recursos); **Servicios de Sincronización y Coordinación** para el manejo del tiempo (sincronización de relojes), concurrencia, paralelismo y transacciones; un **Servicio de Gestión de Procesos** que se encargará de la asignación de procesadores, de la planificación interna y global de los procesos, y de la migración de procesos; un **Sistema de Archivos Distribuidos** que permita gestionar los distintos sistemas de archivos en los

diversos nodos y, el compartimiento de información de manera transparente; **Servicios de Comunicación** que permitan la interacción entre los modelos existentes: cliente/servidor, intermediarios (*proxys*, *caches*, etc.), comunicación en grupo, entre otros. Las interacciones se llevan a cabo a través de pase de mensajes, llamadas a procedimientos y métodos remotos (RPC, RMI, etc.), y memorias compartidas, entre otros.

El micronúcleo consta sólo de las funciones absolutamente esenciales del núcleo del sistema operativo; coordina las interacciones entre los procesos servidores (componentes del sistema operativo externos al micronúcleo) a través de mensajes (valida los mensajes y los pasa entre los componentes, entre otras cosas), y otorga el acceso al hardware [44].

3.1.3. Arquitectura del SOW

La arquitectura del SOW que proponemos está fundamentada en el uso de motores de búsqueda y repositorios. Dicha arquitectura consta de cuatro subsistemas [1]: subsistema manejador de repositorios, subsistema manejador de objetos móviles, subsistema manejador de comunidades y subsistema manejador de recursos (ver figura 3.2).

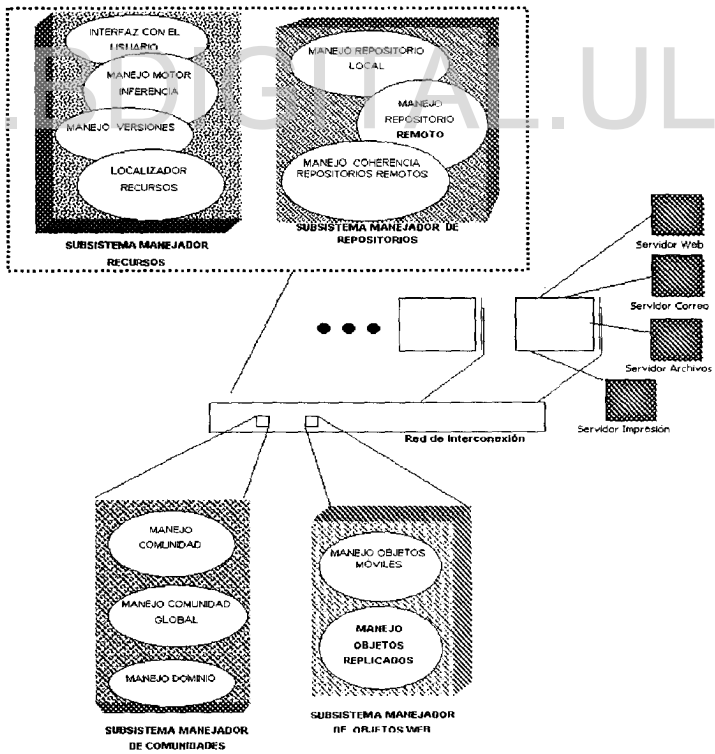

Figura 3.2: Arquitectura del SOW propuesto

- *El subsistema manejador de repositorios* provee técnicas para organizar la información almacenada en los repositorios locales y remotos, también ayuda a coordinar el acceso a los mismos. Cada nodo del SOW posee un repositorio local, el cual usa para almacenar y continuamente actualizar la información acerca de los servicios y recursos disponibles en éste. Por otro lado, a través de los repositorios remotos se realiza el *web caching* del sistema, en él se almacena la información externa más frecuentemente usada, esto con el fin de proveer respuestas más rápidas al usuario. El subsistema manejador de repositorios también debe garantizar la coherencia de la información almacenada.

- *El subsistema manejador de comunidades* establece mecanismos que permiten agrupar eficientemente los nodos de la web en comunidades organizadas. Aquellos nodos que presentan afinidades funcionales y conductuales pueden dinámicamente asociarse entre ellos mismos para formar comunidades. Nuestro SOW tratará a la comunidades desde un punto de vista emergente, es decir, comunidades que emergen y se adaptan a su entorno; se auto-organizan de acuerdo a los requerimientos que surjan; careciendo de un control centralizado.

- *El subsistema manejador de objetos web* provee mecanismos para controlar la migración y la réplica de los objetos del sistema en los diferentes dominios de la red. A través del manejo de objetos web se puede reducir el número de comunicaciones, y mejorar la tolerancia a fallas del sistema. Para el manejo de objetos web se emplea una política de gestión basada en trazas, esto con el fin facilitar la ubicación de los objetos migrados y replicados.

- *El subsistema manejador de recursos* ofrece los medios para la localización y asignación de recursos en el SOW. Este sistema es responsable de actividades tales como el manejo de los motores de búsqueda y el manejo de versiones. Este sistema es el que se desarrolla en este trabajo, y se describe con todos sus detalles en las siguientes secciones.

3.2. Descripción del Sistema Manejador de Recursos (SMR) Propuesto.

A través de nuestra propuesta de Sistema Manejador de Recursos para el SOW se pretende coordinar mecanismos que permitan administrar e integrar recursos computacionales presentes en la Web. Se considera un recurso computacional Web a todo lo que pueda ser manipulado desde un nodo del SOW. Los recursos pueden tener representación física, como por ejemplo, un archivo o una impresora, también representaciones abstractas, como el tiempo de CPU. Debido a la naturaleza heterogénea de los recursos en la Web, es imposible crear un patrón de acceso común para todos los recursos, sin embargo, es posible realizar clasificación de recursos y establecer un conjunto de propiedades que describan a los mismos, así como un conjunto de operaciones atribuibles a ellos. Por ejemplo, una impresora puede ser descrita por propiedades tales como marca, nombre, modelo, tipo (HP LaserJet 5, linus, laser), y por operaciones que ésta pueda realizar, como imprimir un documento *postcript*.

Este esquema de clasificación de recursos se ajusta muy bien al modelo orientado a objetos, donde una jerarquía de clases se puede usar para organizar las diferentes tipos de recursos. Así, podríamos usar un modelo de objetos independiente de la plataforma, que permita organizar un universo heterogéneo de recursos en un conjunto homogéneo de objetos.

El SOW debe proveer de forma transparente los servicios necesarios para atender las solicitudes de los usuarios. De cierto modo, el SOW actuaría como un sistema de meta-computación, pero hay dos cualidades que diferencian nuestra propuesta de SOW de un meta-computador convencional: el *acceso abierto* y la *universalidad*.

Acceso Abierto: La mayoría de los proyectos de meta-computación para el manejo de recursos en sistemas globales, tales como Netsolve [11], Globe [48], Legion [20] y Globus [18]; requieren privilegios de entrada y un catálogo global de recursos. Esto ha sido útil para "redes cerradas", pero resulta poco práctico en el caso de una inmensa red abierta como la Web. En respuesta a esta situación, se propone para el SOW usar en lugar de un catálogo global de recursos, varios repositorios de datos distribuidos en la red, de manera tal que cada nodo del SOW mantenga repositorios que le provean información útil para la ubicación de un recurso. Para esta localización de recursos mediante repositorios, hemos sugerido el uso de "motores de búsqueda", los cuales pueden buscar

por toda la red el recurso solicitado valiéndose de la información suministrada por los repositorios que se encuentran distribuidos en los nodos del SOW.

Universalidad: Pretender administrar los recursos disponibles en la Web, implica afrontar ciertos conflictos de interoperabilidad que surgen debido a la diversidad de plataformas presentes. Un recurso solicitado podría estar presente en diversos sitios de la red, y en cada uno de esos sitios, el recurso estaría soportado por su propio hardware y sistema operativo local, esto abre la posibilidad de tener varias versiones de un recurso distribuidas en la red. En virtud de esta situación, hemos propuesto un enfoque "versionado" del SOW, es decir, que el SOW podrá administrar los recursos en sus distintas versiones.

Tenemos entonces que nuestro SMR se concibe como un sistema distribuido versionado apoyado en un diseño basado en el uso de motores de búsqueda. El motor de búsqueda es el componente principal de la arquitectura del SOW. Todos los requerimientos hechos al sistema son manejados por el motor de búsqueda. Éste funciona como un sistema reactivo de manejo de demandas a través del cual se administran los recursos del ambiente computacional heterogéneo provisto por la Web.

El SOW posee varios motores de búsqueda distribuidos en la red. Cuando un usuario intenta tener acceso a un recurso de la web, el SOW recibe la petición, dicha petición es manejada por el motor de búsqueda del nodo local, el cual consulta a sus repositorios para determinar si es posible satisfacer la petición localmente, de no ser así, la petición es enviada a otro motor de búsqueda y se repite el proceso anterior hasta conseguir satisfacer la petición. El motor de búsqueda está en capacidad de manejar peticiones provenientes desde un usuario o desde otro motor de búsqueda.

La información requerida por cada motor de búsqueda para el manejo de las peticiones se encuentra almacenada en sus repositorios. Cada motor de búsqueda tiene acceso a un repositorio de recursos locales y a un repositorio de recursos remotos. El repositorio de recursos locales siempre es consultado, en cambio el repositorio de recursos remotos sólo es consultado cuando no se puede satisfacer una petición de recursos a nivel local. En el SOW, el SMR será responsable de:

Manejo de requerimiento para procesar las peticiones que llegan al sistema.

Manejo de versiones para determinar las versiones de servicios necesarias para procesar un requerimiento.

Asignación de recursos para definir y designar los recursos que serán destinados para satisfacer un requerimiento.

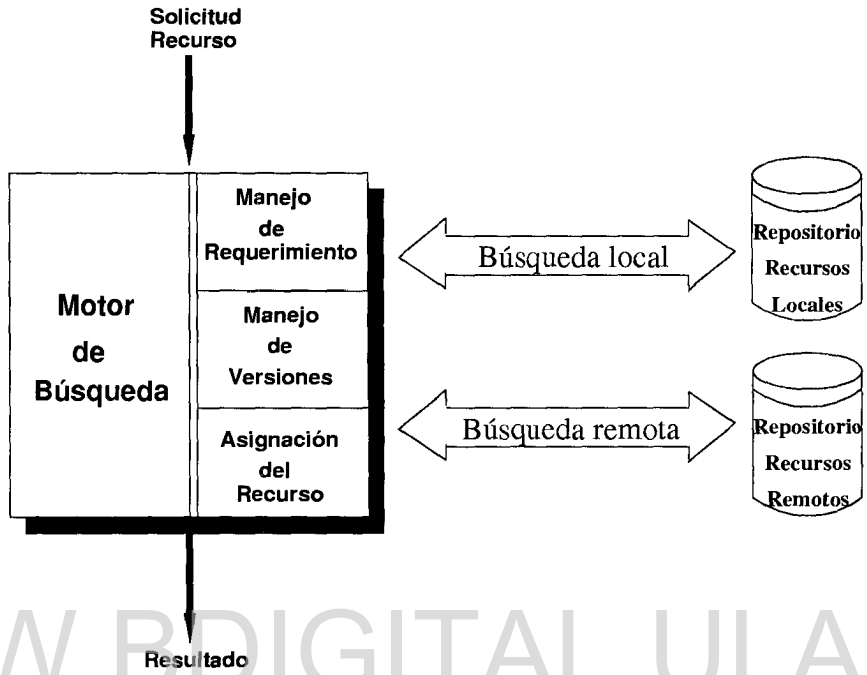

Figura 3.3: Actividad del motor de búsqueda

Capítulo 4

Aspectos del Diseño del SMR

4.1. Arquitectura del SMR

En el Sistema Manejador de Recursos (SMR) del SOW las operaciones son coordinadas por cuatro componentes fundamentales, éstos son: la interfaz con el usuario, el sistema de razonamiento, el localizador de servicios, y el manejador de versiones. Estos cuatro componentes constituyen las unidades funcionales básicas de la arquitectura del SMR. A continuación se presenta una breve descripción de cada una de estas unidades (en la figura 4.1 se muestra la arquitectura del SMR).

Interfaz con el usuario. La unidad de *interfaz con el usuario* se encarga de recibir solicitudes de los usuarios, enviarlas al sistema de razonamiento, y presentar los resultados de las solicitudes a los usuarios. En esta unidad se validan las entradas al sistema, y se analiza la sintaxis y la semántica de la información ingresada.

Sistema de razonamiento. Es la unidad corazón del sistema. En esta unidad se procesan los requerimientos recibidos de la *interfaz con el usuario*, con el fin de dar una respuesta adecuada a dichos requerimientos. Esta unidad posee un motor de inferencia capaz de coordinar inteligentemente los mecanismos de localización, configuración y asignación de servicios requeridos por los usuarios.

Localizador de servicios. La unidad *localizadora de servicios* tiene como objetivo la búsqueda de los servicios y versiones que son requeridas por el *sistema de razonamiento*. A través de un módulo de descubrimiento de servicios se obtiene la información relacionada con la ubicación física de los servicios y de las versiones requeridas. Para ello, la

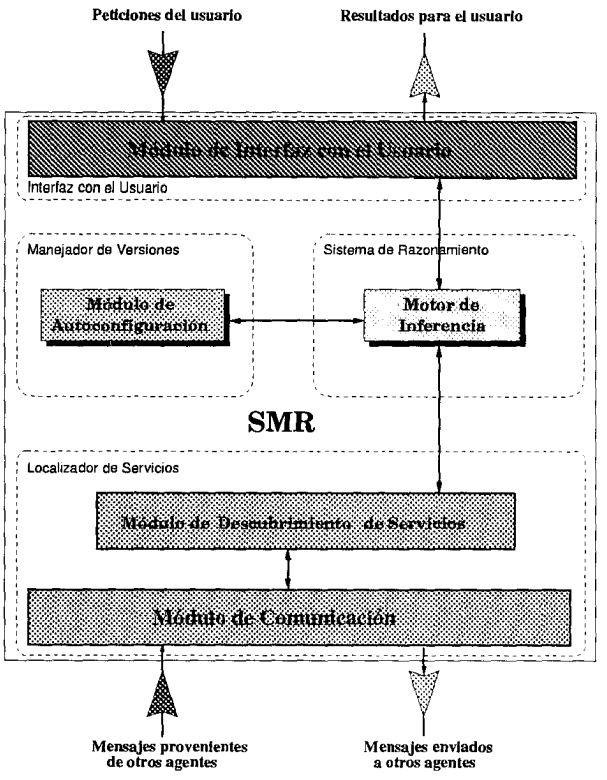

Figura 4.1: Arquitectura del SMR

unidad localizadora de servicios hace uso del módulo de comunicaciones, el cual se encarga de enviar y recibir mensajes hacia y desde otros componentes externos del SMR.

Manejador de versiones. Esta unidad posee un módulo de autoconfiguración, el cual se encarga del manejo de las versiones de los servicios en función de la información provista por el *motor de inferencia*. Esta unidad tiene como resultado final la configuración de un servicio determinado de acuerdo al requerimiento del usuario.

4.2. Diseño del SMR basado en Sistemas Multiagente

Para efectos de diseño, debemos tomar en cuenta las siguientes características del SMR:

- Es el único subsistema del SOW que realiza operaciones a nivel de usuario.
- Provee los mecanismos que permite a los usuarios la manipulación y el acceso a los recursos del sistema de forma confiable y transparente.

- Incluye mecanismos de razonamiento para localización, configuración y asignación de los servicios asociados a los recursos.
- Soporta accesos a un simple nodo o a múltiples nodos para la localización de los recursos.
- Todas sus operaciones se ajustan a un modelo computacional inherentemente distribuido.

Tenemos, entonces, que el SMR se presenta como un sistema distribuido donde sus unidades funcionales interactúan de forma cooperativa para alcanzar los objetivos del sistema. Así, el SMR puede verse como un sistema distribuido compuesto por agentes capaces de cooperar para obtener la solución a problemas relacionados con el manejo de recursos. El resto de este capítulo está dedicado a presentar en detalle aspectos de diseño del SMR bajo un enfoque orientado a Sistemas Multiagente.

4.3. Análisis y Diseño del SMR

Para efectuar el análisis y diseño del sistema, nos apoyamos en la metodología de desarrollo de Sistemas Multiagente MAS-*Common*KADS extendido, la cual se describió en el capítulo 2. Siguiendo el modelo de ciclo de vida de esta metodología, obtenemos el diseño del sistema después de desarrollar las fases de conceptuación y análisis.

4.4. Conceptuación

La conceptuación consiste en la extracción y adquisición de conocimiento para obtener una primera descripción del problema. En esta fase se concibe el problema que se va a resolver y se elabora un primer esbozo del sistema que puede resolverlo. Es una fase informal de toma de contacto con el problema, en la cual se sigue un análisis centrado en el usuario, donde las técnicas de casos de uso y la identificación de actores son de gran utilidad. Se emplea la técnica de casos de usos propuesta por Jacobson [24] para ayudar a entender los requisitos informales [40].

4.4.1. Descripción de actores

En nuestra etapa de conceptuación, un caso de uso corresponde a la descripción de acciones necesarias para producir un resultado útil para un actor. Un actor representa un rol desempeñado por alguna persona, una pieza de software u otro sistema que interactúa con el nuestro. Cada una de las unidades funcionales del SMR descritas en la sección 4.1 desempeñan un rol fundamental para la solución de los problemas, tenemos entonces un actor asociado a cada unidad. Así, definimos al actor *Solicitante*, el cual desempeña roles propios de la unidad *interfaz con el usuario*; el actor *Administrador*, el cual desempeña roles propios de la unidad *motor de inferencia*; el actor *Localizador*, cuyas actividades están relacionadas con la unidad *localizador de servicios*; y el actor *Configurador* , el cual desempeña roles propios de la unidad *manejador de versiones*. En el cuadro 4.1 se identifican y describen los actores, y se identifican los casos de uso asociados a cada uno de los actores.

Actor	**Descripción**	**Casos de Uso**
Solicitante	Recibe peticiones y emite los resultados al usuario, permite una acción cooperativa entre el usuario y el SMR.	Procesar solicitud; Armar respuesta.
Administrador	Procesa los requerimientos provenientes de la interfaz de usuario. Coordina con otros actores mecanismos de localización, configuración y asignación de servicios requeridos.	Coordinar petición; Caracterizar versiones; Obtener configuración; Asignar servicio; Ejecutar servicio.
Localizador	Provee técnicas para la localización de servicios y sus respectivas versiones. Realiza consultas a repositorios y se comunica con actores remotos	Descubrir servicio; Consulta local; Consulta Remota; Comunicarse con actores remotos.
Configurador	Provee mecanismos para configuración dinámica de servicios.	Configurar Servicios.

Cuadro 4.1: Actores y Casos de Uso

4.4.2. Descripción de los casos de uso

El resto de la fase de conceptuación es la descripción de los diferentes casos de uso de cada uno de los actores. Para la descripción de los casos de uso, utilizaremos la notación textual

propuesta por Rumbaugh [42], donde los casos de uso se describen a través de plantillas. A continuación se presentan cada una de las plantillas que describen nuestros casos de uso.

4.4.2.1. Casos de Uso del Actor Solicitante

Caso de Uso: *Procesar Solicitud*	1.1
Resumen: El actor *Solicitante* recibe la información dada por el usuario al realizar un requerimiento al SOW, esta información es interpretada y validada para luego iniciar la búsqueda del servicio solicitado. **Actores:** *Solicitante.* **Precondición:** Haber recibido una petición del usuario. **Excepción:** No encontrar información relacionada con la petición. El usuario introdujo una información errónea o incompleta.	

Cuadro 4.2: Caso de uso *Procesar Solicitud* para el actor *Solicitante*

Caso de Uso: *Armar Respuesta*	1.2
Resumen: El actor *Solicitante* recibe información del actor *Administrador* como respuesta a un requerimiento realizado por el usuario. De esta información se seleccionan los parámetros necesarios para estructurar una respuesta en un formato legible para el usuario. **Actores:** *Solicitante* y *Administrador.* **Precondición:** Haber obtenido respuesta del actor *Administrador.* **Excepción:** El actor Administrador no puede satisfacer la petición.	

Cuadro 4.3: Caso de uso *Armar Respuesta* para el actor *Solicitante*

4.4.2.2. Casos de Uso del Actor Administrador

Caso de Uso: *Coordinar Petición* 2.1

Resumen: El actor *Administrador* recibe una petición de servicio del actor *Solicitante*. La información recibida es analizada para fijar los parámetros necesarios para iniciar una búsqueda de servicio y construir la estructura de datos usada para iniciar las tareas de descubrimiento y configuración del servicio.
Actores: *Solicitante* y *Administrador*.
Precondición: Recibir una solicitud del actor *Solicitante*.
Excepción: Se recibe información errónea o incompleta.

Cuadro 4.4: Caso de uso *Coordinar Petición* para el actor *Administrador*

Caso de Uso: *Caracterizar Versiones* 2.2

Resumen: El actor *Administrador* solicita al actor *Localizador* información acerca de un servicio determinado. El actor *Administrador* recibe como respuesta la ubicación física del servicio requerido y de sus correspondientes versiones. Esta información es estructurada para ser enviada posteriormente al actor *Configurador*.
Actores: *Localizador* y *Administrador*.
Precondición: Obtener como respuesta del actor *Localizador*, la ubicación del servicio y sus respectiva versiones.
Excepción: ¬Precondición.

Cuadro 4.5: Caso de uso *Caracterizar Versiones* para el actor *Administrador*

Caso de Uso: *Obtener Configuración* 2.3

Resumen: El actor *Administrador* recibe respuesta a una solicitud de configuración de servicios, obteniendo el servicio configurado según el requerimiento del usuario.
Actores: *Configurador* y *Administrador*.
Precondición: Haber hecho una solicitud de configuración de servicio.
Excepción: La configuración del servicio no tuvo éxito.

Cuadro 4.6: Caso de uso *Obtener Configuración* para el actor *Administrador*

Caso de Uso: *Asignar Servicio*

2.4

Resumen: El actor *Administrador* reserva un servicio para ser usado por el usuario que lo requiere. Se designa el hardware y el software que será usado para ejecutar el servicio.
Actores: *Administrador.*
Precondición: Haber obtenido la configuración del servicio a asignar.
Excepción: La configuración del servicio no tuvo éxito.

Cuadro 4.7: Caso de uso *Obtener Configuración* para el actor *Administrador*

Caso de Uso: *Ejecutar Servicio*

2.5

Resumen: El actor Administrador realiza una llamada al servicio requerido. El servicio es reservado hasta completar su ejecución. Los resultados de la ejecución son devueltos al actor solicitante.
Actores: *Solicitante*, *Administrador* y servicios.
Precondición: Tener la configuración y la ubicación del servicio.
Excepción: ¬Precondición, no se pudo ejecutar el servicio.

Cuadro 4.8: Caso de uso *Ejecutar Servicio* para el actor *Administrador*

4.4.2.3. Casos de Uso del Actor Localizador

Caso de Uso: *Descubrir Servicio*	3.1
Resumen: El actor *Localizador*, al recibir una solicitud de búsqueda de servicio, se comunica con los Sistemas Manejador de Repositorios y de Comunidades para obtener información relacionada con la ubicación del servicio y de la versión requerida. De este caso de uso se derivan tres "sub-casos de uso": *Consulta Local*, *Consulta Remota* y *Consultar Actores Remotos* **Actores:** *Localizador*, Sistema Manejador de Repositorios y Sistema Manejador de Comunidades. **Precondición:** Haber recibido una solicitud de búsqueda de servicio. **Excepción:** No se puede establecer comunicación con el Sistema Manejador de Repositorios o de Comunidades.	

Cuadro 4.9: Caso de uso *Descubrir Servicio* para el actor *Localizador*

Caso de Uso: *Consulta Local*	3.1.1
Resumen: El actor *Localizador* se comunica con el repositorio de recursos locales del Sistema Manejador de Repositorios para buscar el servicio en la red local. **Actores:** *Localizador* y Sistema Manejador de Repositorios. **Precondición:** Haber recibido una solicitud de búsqueda de servicio. **Excepción:** No se puede establecer comunicación con el Sistema Manejador de Repositorios.	

Cuadro 4.10: Sub-caso de uso *Consulta Local* para el actor *Localizador*

Caso de Uso: *Consulta Remota*	3.1.2
Resumen: El actor *Localizador* se comunica con el repositorio de recursos remotos del Sistema Manejador de Repositorios para realizar la búsqueda del servicio requerido en sitios remotos. **Actores:** *Localizador*, Sistema Manejador de Repositorios. **Precondición:** Haber recibido una solicitud de búsqueda de servicio. **Excepción:** No se puede establecer comunicación con el Sistema Manejador de Repositorios.	

Cuadro 4.11: Sub-caso de uso *Consulta Remota* para el actor *Localizador*

Caso de Uso: *Consultar Actores Remotos*	3.1.3
Resumen: El actor *Localizador* consulta actores *administradores* de otros dominios administrativos para intentar localizar un servicio remoto que no pudo ser encontrado por el agente *Localizador* local. Se usa para ello al Sistema Manejador de Comunidades. **Actores:** *Localizador*, Sistema Manejador de Comunidades. **Precondición:** El requerimiento de servicio no se pudo satisfacer localmente. **Excepción:** No se puede establecer comunicación con el Sistema Manejador de Comunidades.	

Cuadro 4.12: Sub-caso de uso *Consultar Actores Remotos* para el actor *Localizador*

Caso de Uso: *Configurar Servicios*	4.1
Resumen: El actor *Configurador* prepara el servicio requerido para ser usado. Construye la versión de servicio requerida a partir de la información suministrada por el actor *Administrador*. **Actores:** *Configurador*. **Precondición:** Poseer información acerca de las versiones del servicio requerido. **Excepción:** No poder configurar el servicio exitosamente.	

Cuadro 4.13: Caso de uso *Configurar Servicios* para el actor *Configurador*

4.5. Análisis

La etapa de análisis corresponde a la segunda fase del modelo de ciclo de vida de la metodología MAS-*Kommon*KADS. En esta parte aplicamos los modelos de *agente*, *tareas*, *organización*, *comunicación*, *coordinación*, e *inteligencia*, para obtener las especificaciones y requerimientos del Sistema Multiagente.

4.5.1. Modelo de Agente

Este es el modelo central de la metodología. El propósito del modelo de agente es describir los agentes que participan en la resolución del problema. El modelo de agente recoge las características de los agentes del sistema y sirve de puente con el resto de los modelos [21]

4.5.1.1. Identificación de los Agentes

La identificación de los agentes se realiza en base a los actores definidos en la fase de conceptuación. Según los roles definidos durante esta fase, tenemos que el SMR provee las siguiente funcionalidades: *demanda de recursos*, *procesamiento de las demandas*, *localización de servicios*, y *configuración de servicios*. Estas funciones representan los roles de cada uno de los actores identificados en la fase de conceptuación. Vamos a mantener a todos estos actores como agentes de nuestro sistema. Entonces, podemos identificar cuatro agentes, a los cuales hemos denominado:

1. Agente solicitante.
2. Agente administrador.
3. Agente localizador.
4. Agente de configuración.

Estos son los agentes que constituyen nuestro Modelo de Agente. La siguiente sección está dedicada a examinar en detalle las características de cada uno de los agentes: sus objetivos, capacidades de razonamiento, habilidades, restricciones, servicios, etc.

4.5.1.2. Estructura del Modelo de Agente

La metodología MAS-*Kommon*KADS provee plantillas textuales a través de las cuales se describe la estructura del modelo de agentes. A continuación incluimos una serie de plantillas textuales que describen cada uno de los constituyentes del modelo de agente del SMR.

Agente Solicitante

Agente: *Solicitante*

1.1

Nombre: Solicitante.
Tipo: Agente software: agente de interfaz (entradas y salidas).
Papel: Interacción entre el usuario y el sistema.
Posición: Es el agente de más alto nivel dentro del sistema multiagente.
Descripción: Este agente es responsable de la comunicación entre el usuario y el sistema, prepara los requerimientos del usuario para ser entregados al agente *Administrador* y recibe respuestas de éste para ser entregadas al usuario. Este agente valida la estructura semántica y sintáctica de los requerimientos de usuario.

Cuadro 4.14: Agente *Solicitante*

Objetivo - *Agente Solicitante*

1.2

Nombre: Comunicación hombre-máquina a alto nivel.
Tipo: Objetivo persistente.
Parámetros de entrada: Datos del requerimiento de usuario.
Parámetros de salida: Solución al requerimiento.
Condición de activación: Recibir un requerimiento del usuario, recibir una respuesta del agente *Administrador*.
Condición de finalización: Solución al requerimiento del usuario.
Condición de éxito: Se obtuvo una respuesta al requerimiento del usuario.
Condición de fracaso: ¬Condición de éxito .
Lenguaje de representación: Lenguaje natural.
Ontología: Ontología de servicios SMR, Ontología de interfaz.
Descripción: El agente *Solicitante* tiene como objetivo proveer mecanismos que permitan una comunicación a alto nivel con el usuario del sistema. Esta comunicación se inicia con una solicitud de requerimiento por parte del usuario y finaliza con una respuesta de éxito o fracaso como solución al requerimiento.

Cuadro 4.15: Objetivo del agente *Solicitante*

Servicio - *Agente Solicitante*

1.3

Nombre: procesar-petición.
Tipo: Gratuito, concurrente.
Parámetros de entrada: Nombre del servicio SOW requerido: nombre.
Parámetros de salida: Resultados y salida producto de la ejecución del servicio SOW requerido.
Lenguaje de representación: Lenguaje natural.
Ontología: Ontología de servicios SMR.

Cuadro 4.16: Servicio del agente *Agente Solicitante*

Propiedad-Servicio - *Agente Solicitante*

1.4

Propiedades: Complejidad del Servicio.
Calidad del Servicio.

Cuadro 4.17: Propiedad-servicio del agente *Solicitante*

Propiedad-Servicio - *Agente Solicitante*

1.4.1

Nombre: Complejidad.
Valor: Estándar, avanzada.
Descripción: Esta propiedad está relacionada con el nivel de complejidad de los servicio que provee el agente *Solicitante*. Éstos servicios están orientados a dos tipos de usuarios: 1. *usuario estándar*, el cual provee la información mínima para la solicitud de un requerimiento; 2. *usuario avanzado*, el cual está en capacidad de proveer datos adicionales para la solicitud de un requerimiento, tales como opciones avanzadas o preferencias personales.

Cuadro 4.18: Propiedad-servicio complejidad del agente *Solicitante*

Propiedad-Servicio - *Agente Solicitante*

1.4.2

Nombre: Calidad.
Valor: Mala, buena, excelente.
Descripción: Esta propiedad se refiere al nivel de eficiencia alcanzado luego de la ejecución del servicio *procesar-petición*.

Cuadro 4.19: Propiedad-servicio calidad del agente *Solicitante*

Capacidad General - *Agente Solicitante*

1.5

Habilidades: Alimentar al resto del sistema multiagente con información proveniente del ambiente del usuario.
Lenguaje de Representación: Lenguaje natural.

Cuadro 4.20: Capacidad general del agente *Solicitante*

Restricción - *Agente Solicitante*

1.6

Normas: Una vez que se ha iniciado el procesamiento de un requerimiento, el agente *Solicitante* debe esperar siempre por una respuesta del agente *Administrador*, ya que es éste último quien toma las decisiones.
Preferencias: Los requerimientos que llegan al agente *Solicitante* son procesados en orden de llegada (el sistema soporta manejo de prioridades, pero esta tarea le corresponde al agente *Administrador*).
Permisos: Sólo el agente *Administrador* tiene acceso al agente *Solicitante*.

Cuadro 4.21: Restricción del agente *Solicitante*

Agente Administrador

Agente: *Administrador*

2.1

Nombre: *Administrador.*
Tipo: Agente software: agente proactivo.
Papel: Manejo de demandas del sistema.
Posición: Es el agente central del sistema, coordina actividades con el resto de los agentes.
Descripción: Este agente es el encargado del manejo de todos los requerimientos que llegan al sistema, coordina las actividades de búsqueda, configuración y asignación de los recursos solicitados por el usuario.

Cuadro 4.22: Agente *Administrador*

Objetivo: *Agente Administrador*

2.2

Nombre: Manejar requerimientos.
Tipo: Objetivo no-persistente.
Parámetros de entrada: Requerimiento a procesar.
Parámetros de salida: Resultado de la ejecución del servicio requerido.
Condición de activación: Recibir un requerimiento de servicio, recibir un resultado de búsqueda de servicio, recibir un resultado de la configuración de un servicio.
Condición de finalización: Asignación y ejecución del servicio requerido.
Condición de éxito: Se logró localizar, configurar y asignar el servicio requerido.
Condición de fracaso: ¬Condición de éxito .
Lenguaje de representación: Lenguaje natural.
Ontología: Ontología de servicios SMR.
Descripción: El agente *Administrador* tiene como objetivo proveer mecanismos que permitan satisfacer las demandas de servicio que llegan al sistema. Para lograr dicho objetivo, este agente trabaja en forma coordinada con el resto de los agentes del sistema y emplea mecanismos de razonamiento para la toma de decisiones.

Cuadro 4.23: Objetivo del agente *Administrador*

Servicio - *Agente Administrador*	2.3.1
Nombre: asignar-servicio. **Tipo:** Gratuito, concurrente. **Parámetros de entrada:** Nombre del servicio SOW requerido. **Parámetros de salida:** dirección del sitio donde es asignado el servicio SOW requerido. Parámetros de excepciones, tales como: el servicio no fue encontrado, la versión no fue encontrada, no se pudo configurar el servicio. **Lenguaje de representación:** Lenguaje natural. **Ontología:** Ontología de servicios SMR.	

Cuadro 4.24: Servicio del agente *Administrador*

Servicio - *Agente Administrador*	2.3.2
Nombre: ejecutar-servicio. **Tipo:** Gratuito, concurrente. **Parámetros de entrada:** Nombre y dirección del servicio SOW requerido. **Parámetros de salida:** Resultados y salida producto de la ejecución del servicio SOW requerido. Parámetros de excepciones, tales como: error en la ejecución del servicio, etc. **Lenguaje de representación:** Lenguaje natural. **Ontología:** Ontología de servicios SMR.	

Cuadro 4.25: Servicio del agente *Administrador*

Propiedad-Servicio - *Agente Administrador*	2.4
Propiedades: Complejidad del Servicio. Desempeño del Servicio.	

Cuadro 4.26: Propiedad-servicio del agente *Administrador*

Propiedad-Servicio - *Agente Administrador* 2.4.1

Nombre: Complejidad.
Valor: Estándar, avanzada.
Descripción: Esta propiedad está relacionada con el nivel de complejidad de los servicio que provee el agente *Administrador*. En el nivel estándar, el agente maneja información básica para satisfacer el servicio requerido. En el nivel avanzado, el agente maneja información adicional, la cual es aprovechada para poder satisfacer el servicio requerido de forma más eficiente.

Cuadro 4.27: Propiedad-servicio *Complejidad* del agente *Administrador*

Propiedad-Servicio - *Agente Administrador* 2.4.2

Nombre: Desempeño.
Valor: Malo, bueno, excelente.
Descripción: Esta propiedad se refiere al tiempo de respuesta alcanzado luego de realizar todas las actividades necesarias para satisfacer el servicio requerido.

Cuadro 4.28: Propiedad del servicio *Desempeño* del agente *Administrador*

Capacidad General - *Agente Administrador* 2.5

Habilidades: Posee capacidades de razonamiento, puede evaluar diferentes alternativas de solución , además de negociar con otros agentes para seleccionar la mejor opción.
Lenguaje de Representación: Lenguaje natural.

Cuadro 4.29: Capacidad general del agente *Administrador*

Restricción - *Agente Administrador*

2.6

Normas: El agente *Administrador* debe activar mecanismos que le permitan percibir cambios en su entorno, con el fin de confirmar la llegada de información proveniente de algún agente externo.
Preferencias: Se busca en lo posible satisfacer los requerimiento de servicios a nivel local.
Permisos: Todos los agentes del sistema tienen acceso al agente *Administrador*.

Cuadro 4.30: Restricción del agente *Administrador*

Agente Localizador

Agente: *Localizador*

3.1

Nombre: *Localizador*.
Tipo: Agente software: agente de comunicación.
Papel: Comunicación con otros agentes para la localización de un recurso determinado.
Posición: Es el agente de más bajo nivel dentro del sistema multiagente.
Descripción: Este agente es el encargado del búsqueda de los recursos, a través de él se realizan consultas a agentes repositorios de datos y a otros agentes remotos, los cuales proveen información útil para la localización del recurso.

Cuadro 4.31: Agente *Localizador*

Objetivo - *Agente Localizador*

3.2

Nombre: Localización de un servicio determinado.
Tipo: Objetivo persistente.
Parámetros de entrada: Servicio, versión.
Parámetros de salida: Ubicación de servicio.
Condición de activación: Recibir una solicitud de búsqueda del agente *Administrador*.
Condición de finalización: Localización del servicio, excepción en la búsqueda.
Condición de éxito: Se obtiene información de la ubicación del servicio buscado.
Condición de fracaso: ¬Condición de éxito, se sobrepasa el valor del TTL (*time to live*) permitido.
Lenguaje de representación: Lenguaje natural.
Ontología: Ontología de servicios SMR, ontología de comunicación del SOW.
Descripción: El agente *Localizador* tiene como objetivo proveer mecanismos que permitan la localización de un recurso con el fin de satisfacer una petición de usuario. La búsqueda se realiza a través de comunicaciones con agentes repositorios de datos o con agentes *Administradores* remotos. Los primeros proveen información útil para la búsqueda de los recursos, mientras que los segundos ayudan en la tarea de búsqueda de recursos remotos.

Cuadro 4.32: Objetivo del agente *Localizador*

Servicio - *Agente Localizador*

3.3

Nombre: descubrir-servicio.
Tipo: Gratuito, concurrente.
Parámetros de entrada: Datos del servicio requerido: nombre, id, versión.
Parámetros de salida: Dirección del servicio encontrado. Parámetros de excepciones tales como: el servicio no fue encontrado, error en la comunicación, etc.
Lenguaje de representación: Lenguaje natural.
Ontología: Ontología de servicios SMR, ontología de comunicación del SOW.

Cuadro 4.33: Servicio del agente *Localizador*

Propiedad-Servicio - *Agente Administrador*

3.4

Propiedades: Calidad del Servicio.
Confiabilidad del Servicio.

Cuadro 4.34: Propiedad-servicio del agente *Administrador*

Propiedad-Servicio - *Agente Localizador*

3.4.1

Nombre: Calidad.
Valor: Mala, buena, excelente.
Descripción: Esta propiedad se refiere al nivel de eficiencia alcanzado luego de realizar todas las actividades necesarias para localizar un servicio determinado.

Cuadro 4.35: Propiedad-servicio *Calidad* del agente *Localizador*

Propiedad-Servicio - *Agente Localizador*

3.4.2

Nombre: Confiabilidad.
Valor: Confiable, no-confiable.
Descripción: Esta propiedad se refiere a la habilidad que tiene el agente *Localizador* de responder consistentemente.

Cuadro 4.36: Propiedad-servicio *Confiabilidad* del agente *Localizador*

Capacidad General - *Agente Localizador*

3.5

Habilidades: Posee capacidades para coordinar la comunicación con agentes externos y agentes remotos. Posee capacidades para realizar búsquedas inteligentes.
Lenguaje de Representación: Lenguaje natural.

Cuadro 4.37: Capacidad general del agente *Localizador*

Restricción - *Agente Localizador*

3.6

Normas: El agente *Localizador* debe activar mecanismos que le permitan una comunicación efectiva con otros agentes. El agente *Localizador* siempre se comunica con el Sistema Manejador de Repositorios, quien ayuda en la búsqueda de los servicios, si esta ayuda falla se recurre al Sistema Manejador de Comunidades para canalizar la búsqueda por medio de agentes remotos.
Preferencias: Se busca en lo posible satisfacer los requerimiento de servicios a nivel local.
Permisos: El agente *Administrador* es el único agente del SMR que tiene acceso al agente *Localizador*. Otros agentes externos como los agentes del Sistema Manejador de Repositorios y del Sistema Manejador de Comunidades también tienen acceso al agente *Localizador*.

Cuadro 4.38: Restricción del agente *Localizador*

Agente De Configuración

Agente: *De Configuración*

4.1

Nombre: *De Configuración.*
Tipo: Agente software: agente reactivo.
Papel: Configuración de servicios.
Posición: Es un agente de tareas específicas que colabora con el agente *Administrador.*
Descripción: Este agente es el encargado del manejo de las versiones de los servicios, para ello interpreta la información que le suministra el agente *Administrador*, con el fin de configurar una solución acorde con el requerimiento del usuario.

Cuadro 4.39: Agente *de Configuración*

Objetivo: *Agente de Configuración*

4.2

Nombre: Configurar Servicios.
Tipo: Objetivo persistente.
Parámetros de entrada: Datos del servicio a configurar.
Parámetros de salida: Parámetros del servicio configurado.
Condición de activación: Recibir una solicitud de configuración de un servicio.
Condición de finalización: Asignación de parámetros de configuración del servicio.
Condición de éxito: Se logró configurar el servicio requerido.
Condición de fracaso: La información suministrada es insuficiente para una configuración exitosa.
Lenguaje de representación: Lenguaje natural.
Ontología: Ontología de servicios SMR.
Descripción: El agente *de Configuración* tiene como objetivo proveer mecanismos que permitan la autoconfiguración de servicios en función de la información provista por el agente *Administrador*. De esta manera, el servicio queda listo para ser usado.

Cuadro 4.40: Objetivo del agente *de Configuración*

Servicio - *Agente de Configuración*

4.3

Nombre: configurar-servicios-SOW.
Tipo: Gratuito, concurrente.
Parámetros de entrada: Datos del servicio a configurar: nombre, localización, versiones.
Parámetros de salida: Parámetros del servicio configurado: id-servicio, localización y versión para la ejecución adecuada del servicio.
Lenguaje de representación: Lenguaje natural.
Ontología: Ontología de servicios SMR.

Cuadro 4.41: Servicio del agente *de Configuración*

Propiedad-Servicio - *Agente de Configuración*

4.4

Propiedades: Calidad del Servicio.
Confiabilidad del Servicio.

Cuadro 4.42: Propiedad-servicio del agente *de Configuración*

Propiedad-Servicio - *Agente de Configuración*

4.4.1

Nombre: Calidad.
Valor: Mala, buena, excelente.
Descripción: Esta propiedad se refiere al nivel de eficiencia alcanzado luego de realizar todas las actividades necesarias para la configuración de un servicio determinado.

Cuadro 4.43: Propiedad-servicio *Calidad* del agente *de Configuración*

Propiedad-Servicio - *Agente de Configuración*

4.4.2

Nombre: Confiabilidad.
Valor: Confiable, no-confiable.
Descripción: Esta propiedad se refiere a la habilidad que tiene el agente de *Configuración* de responder consistentemente.

Cuadro 4.44: Propiedad-servicio *Confiabilidad* del agente de *Configuración*

Capacidad General - *Agente de Configuración*

4.5

Habilidades: Proporciona medios para lograr una eficiente ejecución del servicio requerido.
Lenguaje de Representación: Lenguaje natural.

Cuadro 4.45: Capacidad general del agente *de Configuración*

Restricción - *Agente de Configuración* 4.6
Normas: Las acciones de este agente están supeditadas por el agente *Administrador*. El agente *de Configuración* se debe limitar a interpretar las conclusiones del agente *Administrador* con el fin de configurar una solución acorde a los requerimientos. **Preferencias:** No aplica. **Permisos:** Sólo el agente *Administrador* tiene acceso al agente *de Configuración*.

Cuadro 4.46: Restricción del agente *de Configuración*

4.5.2. Modelo de Tareas

A través de este modelo se describen las tareas que realizan los agentes dentro del sistema para alcanzar sus objetivos.

4.5.2.1. Identificación de las Tareas

La identificación de las tareas del SMR se deriva de los actores y casos de usos definidos durante la fase de conceptuación. Las tareas identificadas se enumeran en el cuadro 4.47

Tareas del SMR	
1. Tareas de solicitud	1.1. Procesar Solicitud. 1.2. Armar Respuesta.
2. Tareas de administración	2.1. Coordinar Solicitud. 2.2. Caracterizar Versiones. 2.3. Asignar Recurso/Servicio. 2.4. Ejecutar Servicio.
3. Tareas de localización	3.1. Coordinar Búsqueda 3.2. Descubrir Servicio.
4. Tareas de configuración	4.1. Configurar Servicios.

Cuadro 4.47: Tareas del SMR

4.5.2.2. Estructura del Modelo de Tareas

A continuación se incluyen las plantillas textuales que describen la estructura del modelo de tareas del SMR.

4.5.2.2.1. Tareas del Agente Solicitante

Tarea: Procesar Solicitud

Tarea: *Procesar Solicitud*	1.1
Objetivo: Introducir al sistema las peticiones de usuario. **Descripción:** A través de esta tarea el sistema recibe la información dada por el usuario al momento de realizar un requerimiento al SOW, esta información es validada y estructurada para luego ser enviada al agente *Administrador* como una solicitud de búsqueda del recurso requerido. **Precondición:** La información ingresada debe ser coherente y estar completa. **Frecuencia:** Relativa a la entrada.	

Cuadro 4.48: Tarea *Procesar Solicitud*

Ingrediente-Tarea: *Procesar Solicitud*	1.1.1
Nombre: Petición de servicio. **Contenidos:** Nombre, id del servicio, sitio de origen de la solicitud, tipo de solicitud (estándar o avanzada).	

Cuadro 4.49: Ingrediente de la tarea *Procesar Solicitud*

Capacidad-Tarea: *Procesar Solicitud*

1.1.2

Nombre: Notificación.
Descripción: La tarea *Solicitar Recurso* está en capacidad de notificar al sistema la llegada de las peticiones del usuario.

Cuadro 4.50: Capacidad de la tarea *Procesar Solicitud*

Detalle-Tarea: *Procesar Solicitud*

1.1.3

1. *leer parámetros de la solicitud.*
2. *validar entrada*
   2.1. *examinar sintaxis*
   2.2. *examinar semántica*
3. *determinar perfil de usuario*
   3.1. *definir origen de la solicitud*
   3.2. *definir tipo de usuario (estándar o avanzado)*
4. *estructurar mensaje de solicitud*
5. *enviar mensaje de solicitud*

Cuadro 4.51: Detalle de la tarea *Procesar Solicitud*

Tarea: Armar Respuesta

Tarea: *Armar Respuesta*

1.2

Objetivo: Suministrar al usuario información relacionada con la solución a su petición.
Descripción: A través de esta tarea el agente *Solicitante* recibe información del agente *Administrador* como respuesta a un requerimiento realizado por el usuario. Esta información es estructurada para luego ser presentada en una manera legible para el usuario.
Precondición: El sistema ha obtenido una solución a la petición del usuario.
Frecuencia: Relativa a la entrada.

Cuadro 4.52: Tarea *Armar Respuesta*

Ingrediente-Tarea: *Armar Respuesta*

1.2.1

Nombre: Resultado de la petición.
Contenidos: Asignación del recurso solicitado, salida producto de la ejecución de los servicios.

Cuadro 4.53: Ingrediente de la tarea *Armar Respuesta*

Capacidad-Tarea: *Armar Respuesta*

1.2.2

Nombre: Informar.
Descripción: La tarea *Armar Respuesta* tiene la capacidad de procesar la información que le es suministrada al usuario como resultado de una petición.

Cuadro 4.54: Capacidad de la tarea *Armar Respuesta*

Detalle-Tarea: *Armar Respuesta*

1.2.3

1 *recibir mensaje de respuesta*
2. *examinar información de respuesta*
   3.1. *precisar respuestas satisfactorias*
   3.2. *precisar excepciones*
4. *determinar dirección de origen de la solicitud*
5. *estructurar mensaje de salida*
6. *enviar respuesta a la dirección de origen*

Cuadro 4.55: Detalle de la tarea *Armar Respuesta*

4.5.2.2.2. Tareas del Agente Administrador

Tarea: Coordinar Solicitud

Tarea: *Coordinar Solicitud*	2.1
Objetivo: Introducir una petición de usuario para ser procesada por el agente *Administrador*. **Descripción:** Esta tarea "dispara" el agente *Administrador* para procesar una petición de servicio. Se reciben los parámetros de la petición y se construye la estructura de datos que será usada para iniciar las tareas de *Descubrir Servicio*, *Configurar Servicio* y *Ejecutar Servicio.* **Precondición:** Se ha ejecutado la tarea *Procesar Solicitud.* **Frecuencia:** Relativa a la entrada.	

Cuadro 4.56: Tarea *Coordinar Solicitud*

Ingrediente-Tarea: *Coordinar Solicitud*	2.1.1
Nombre: Petición de servicio. **Contenidos:** Nombre, id, origen y versión de servicio.	

Cuadro 4.57: Ingrediente de la tarea *Coordinar Solicitud*

Capacidad-Tarea: *Coordinar Solicitud*	2.1.2
Nombre: Preparar búsqueda . **Descripción:** Con la tarea *Coordinar Solicitud* el agente *Administrador* está en capacidad de procesar la solicitud de servicios. Esta tarea sirve como preparación a la tarea *Descubrir Servicio.*	

Cuadro 4.58: Capacidad de la tarea *Coordinar Solicitud*

Detalle-Tarea: *Coordinar Solicitud*

2.1.3

1. *leer parámetros de la petición.*
2. *especificar tipo de petición (básica o avanzada)*
3. *estructurar mensaje de petición*
4. *enviar mensaje de petición al agente Localizador*
5. *recibir mensaje de respuesta a la petición*

Cuadro 4.59: Detalle de la tarea *Coordinar Solicitud*

Tarea: Caracterizar Versiones

Tarea: *Caracterizar Versiones*

2.2

Objetivo: Realizar acciones que permitan discernir acerca de la información obtenida como producto de una petición de servicio.
Descripción: A través de esta tarea el agente *Administrador* interpreta la información obtenida como resultado de una petición de búsqueda de servicio. Datos tales como, la ubicación de los servicios y las versiones disponibles, son contrastados con el conocimiento manejado por el agente *Administrador*. Con esta tarea se inicia el proceso de análisis de la información suministrada, esto con el fin de definir los datos y las directivas que le serán suministradas al agente de *Configuración*.
Precondición: Haber completado la tarea *Descubrir Servicio*.
Frecuencia: Relativa a la entrada y a las decisiones tomadas por el agente *Administrador*

Cuadro 4.60: Tarea *Caracterizar Versiones*

Ingrediente-Tarea: *Caracterizar Versiones*

2.2.1

Nombre: Ubicación del servicio.
Contenidos: Nombre, id, versiones y sitio de destino del servicio.

Cuadro 4.61: Ingrediente de la tarea *Caracterizar Versiones*

Capacidad-Tarea: *Caracterizar Versiones*

2.2.2

Nombre: Inferir información.
Descripción: A través la tarea *Caracterizar Versiones*, el agente *Administrador* está en capacidad de comparar el conocimiento que él maneja con la información suministrada por el agente *Localizador*, y de esta manera precisar las condiciones necesarias para solicitar la configuración de un servicio.

Cuadro 4.62: Capacidad de la tarea *Caracterizar Versiones*

Detalle-Tarea: *Caracterizar Versiones*

2.2.3

1. *leer resultados de solicitud de servicio*
2. *establecer información a recibir*
   - 2.1. *origen de la solicitud*
   - 2.2. *perfil del usuario*
   - 2.3. *preferencias*
   - 2.4. *opciones avanzadas de solicitud de servicio*
3. *analizar información*
   - 3.1. *comparar resultado de la solicitud con el conocimiento propio*
   - 3.2. *determinar parámetros de configuración*
     - 3.2.1. *servicios*
     - 3.2.2. *versiones*
     - 3.2.3. *plataforma*
4. *estructurar mensaje de solicitud de configuración*
5. *enviar mensaje de solicitud de configuración*
6. *recibir respuesta de solicitud de configuración*

Cuadro 4.63: Detalle de la tarea *Caracterizar Versiones*

Tarea: Asignar Recurso/Servicio

Tarea: *Asignar Recurso/Servicio*	2.3
Objetivo: Reservar un servicio para ser usado por el usuario que lo requiere. **Descripción:** A través de esta tarea el agente *Administrador* designa el hardware, y el software que será usado para ejecutar el servicio requerido por el usuario. **Precondición:** Haber completado la tarea *Configurar Servicio.* **Frecuencia:** Relativa a la entrada. **Procedimiento:** Algoritmo de asignación.	

Cuadro 4.64: Tarea *Asignar Recurso/Servicio*

Ingrediente-Tarea: *Asignar Recurso/Servicio*	2.3.1
Nombre: Algoritmo de asignación. **Contenidos:** Asignación de los recursos/servicios, basada en los resultados de la *configuración de servicio.*	

Cuadro 4.65: Ingrediente de la tarea *Asignar Recurso/Servicio*

Capacidad-Tarea: *Asignar Recurso/Servicio*	2.3.2
Nombre: Seleccionar proveedores de servicios. **Descripción:** A través la tarea *Asignar Recurso/Servicio*, el agente *Administrador* está en capacidad de seleccionar un proveedor de servicio que pueda satisfacer el requerimiento procesado.	

Cuadro 4.66: Capacidad de la tarea *Asignar Recurso/Servicio*

Método-Tarea: *Asignar Recurso/Servicio*

2.3.3

Nombre: asignar-recurso-servicio.
Descripción: El método asignar-recurso-servicio incorpora diversas técnicas o estrategias de asignación para la tarea *Asignar Recurso/Servicio.*

Cuadro 4.67: Método de la tarea *Asignar Recurso/Servicio*

Detalle-Tarea: *Asignar Recurso/Servicio*

2.3.4

1. *leer datos de respuesta a la solicitud de configuración*
2. *determinar posibles proveedores de servicios*
3. *seleccionar algoritmo de asignación de servicio*
3. *hacer la asignación del servicio/recurso*

Cuadro 4.68: Detalle de la tarea *Asignar Recurso/Servicio*

Tarea: Ejecutar Servicio

Tarea: *Ejecutar Servicio*

2.4

Objetivo: Preparar un servicio para su ejecución.
Descripción: Una vez que el agente *Administrador* tiene el servicio configurado según el requerimiento del usuario, se realiza la llamada al servicio para lograr su ejecución. La salida producto de la ejecución del servicio es enviada al agente *Solicitante.*
Precondición: Haber completado la tarea *Configurar Servicio.*
Frecuencia: Relativa a la entrada

Cuadro 4.69: Tarea *Ejecutar Servicio*

Ingrediente-Tarea: *Ejecutar Servicio*

2.4.1

Nombre: Servicio configurado.
Contenidos: nombre, id, versión y sitio destino del servicio.

Cuadro 4.70: Ingrediente de la tarea *Ejecutar Servicio*

Capacidad-Tarea: *Ejecutar Servicio*

2.4.2

Nombre: Llamada al servicio.
Descripción: A través la tarea *Ejecutar Servicio* el agente *Administrador* está en capacidad de realizar una llamada al servicio asignado para ejecutarlo según las especificaciones del requerimiento de usuario.

Cuadro 4.71: Capacidad de la tarea *Ejecutar Servicio*

Detalle-Tarea: *Ejecutar Servicio*

2.4.3

1. *leer datos de respuesta a la solicitud de configuración*
2. *preparar servicio*
   - 2.1. *establecer parámetros de configuración*
   - 2.2. *establecer estrategia de invocación*
     - 2.2.1. *definir ámbito del servicio*
     - 2.2.2. *definir "timeout" o TTL*
     - 2.2.3. *precisar preferencias*
     - 2.2.4. *precisar opciones avanzadas de solicitud de servicio*
3. *estructurar mensaje de invocación de servicio*
4. *enviar mensaje de llamada de servicio*
5. *activar timeout*
6. *recibir respuesta de ejecución de servicio*
7. *examinar respuesta*
   - 8.1. *identificar excepciones*
   - 8.2. *identificar error en ejecución*
   - 8.3. *repetir ejecución*
8. *enviar mensaje de respuesta*

Cuadro 4.72: Detalle de la tarea *Ejecutar Servicio*

4.5.2.2.3. Tareas del Agente Localizador

Tarea: Coordinar Búsqueda

Tarea: *Coordinar Búsqueda*

3.1

Objetivo: Accionar mecanismos para organizar la búsqueda del servicio requerido.
Descripción: Con esta tarea se activa el agente *Localizador* para iniciar el proceso de búsqueda de un servicio determinado. Se reciben los parámetros de la petición y se construye el mensaje que será enviado al Sistema Manejador de Repositorios y al Sistema Manejador de Comunidades.
Precondición: Se ha ejecutado la tarea *Coordinar Solicitud.*
Frecuencia: Relativa a la entrada.

Cuadro 4.73: Tarea *Coordinar Búsqueda*

Ingrediente-Tarea: *Coordinar Búsqueda*

3.1.1

Nombre: Solicitud de búsqueda de servicio.
Contenidos: Nombre, id, origen y versión de servicio.

Cuadro 4.74: Ingrediente de la tarea *Coordinar Búsqueda*

Capacidad-Tarea: *Coordinar Búsqueda*

3.1.2

Nombre: Preparar búsqueda.
Descripción: Con la tarea *Coordinar Búsqueda* el agente *Localizador* está en disposición de iniciar una búsqueda de acuerdo a ciertos requermientos.

Cuadro 4.75: Capacidad de la tarea *Coordinar Búsqueda*

Detalle-Tarea: *Coordinar Búsqueda*

3.1.3

1. *recibir mensaje de petición de búsqueda.*
2. *determinar tipo de búsqueda*
3. *estructurar mensaje de petición de búsqueda*

Cuadro 4.76: Detalle de la tarea *Coordinar Búsqueda*

Tarea: Descubrir Servicio

Tarea: *Descubrir Servicio*

3.2

Objetivo: Obtener información referente a la ubicación del servicio requerido mediante consultas realizadas al Sistema Manejador de Repositorios y/o Sistema Manejador de Comunidades.
Descripción: A través de esta tarea el agente *Localizador* se comunica con el Sistema Manejador de Repositorios y el Sistema Manejador de Comunidades con el fin de obtener información útil para la ubicación del servicio y de la versión requerida.
Precondición: Haber completado la tarea *Coordinar Búsqueda.*
Frecuencia: Relativa a la entrada.
Procedimiento: Algoritmo de búsqueda.

Cuadro 4.77: Tarea *Descubrir Servicio*

Ingrediente-Tarea: *Descubrir Servicio*

3.2.1

Nombre: Dirección física.
Contenidos: Nombre, id, versión y sitio de ubicación del servicio.

Cuadro 4.78: Ingrediente de la tarea *Descubrir Servicio*

Capacidad-Tarea: *Descubrir Servicio*

3.2.2

Nombre: Explorar espacio de búsqueda.
Descripción: A través la tarea *Descubrir Servicio* el agente *Localizador* está en capacidad de definir directrices de búsqueda, las cuales se proveen al Sistema Manejador de Repositorios para ayudar a localizar un servicio.

Cuadro 4.79: Capacidad de la tarea *Descubrir Servicio*

Método-Tarea: *Descubrir Servicio*

3.2.3

Nombre: búsqueda-servicio.
Descripción: El método búsqueda-servicio incorpora diversas técnicas o estrategias de búsqueda para la tarea *Descubrir Servicio.*

Cuadro 4.80: Método de la tarea *Descubrir Servicio*

Detalle-Tarea: *Descubrir Servicio*

3.2.4

1. *leer parámetros de solicitud de búsqueda*
2. *enviar mensaje a repositorio local*
3. *recibir respuesta de repositorio local*
4. *examinar respuesta*
   4.1. *identificar excepciones*
   4.2. *identificar solución (encontrado o no encontrado)*
5. *enviar mensaje a repositorio remoto*
6. *recibir respuesta de repositorio remoto*
7. *examinar respuesta*
   7.1. *identificar excepciones*
   7.2. *identificar solución (encontrado o no encontrado)*
8. *enviar mensaje a sistema manejador de comunidades*
9. *recibir respuesta del sistema manejador de comunidades*
10. *examinar respuesta*
    10.1. *identificar excepciones*
    10.2. *identificar solución (encontrado o no encontrado)*
11. *establecer parámetros de solución a la búsqueda de servicio*

Cuadro 4.81: Detalle de la tarea *Descubrir Servicio*

4.5.2.2.4. Tareas del Agente de Configuración

Tarea: Configurar Servicios

Tarea: *Configurar Servicios*

4.1

Objetivo: Preparar un servicio para ser ejecutado en función de las especificaciones del usuario y de las decisiones tomadas por el agente *Administrador*.
Descripción: A través de esta tarea el agente *de Configuración* prepara el servicio requerido para ser usado. Construye la versión de servicio requerida a partir de la información suministrada por el agente *Administrador*.
Precondición: Haber completado la tarea *Seleccionar Versiones*.
Frecuencia: Relativa a la entrada y a las decisiones tomadas por el agente *Administrador*.

Cuadro 4.82: Tarea *Configurar Servicios*

Ingrediente-Tarea: *Configurar Servicios*

4.1.1

Nombre: Servicio configurado.
Contenidos: Nombre, id, versiones y destinos de los servicios.

Cuadro 4.83: Ingrediente de la tarea *Configurar Servicios*

Capacidad-Tarea: *Configurar Servicios*

4.1.2

Nombre: Interpretar conclusiones.
Descripción: A través la tarea *Configurar Servicios* el agente de *Configuración* está en capacidad de interpretar las conclusiones y decisiones del agente *Administrador* con el fin de configurar una solución acorde a los requerimientos del usuario.

Cuadro 4.84: Capacidad de la tarea *Configurar Servicios*

Detalle-Tarea: *Configurar Servicios*

4.1.3

1. *recibir solicitud de configuración*
2. *examinar información*
   2.1. *precisar servicios*
   2.2. *precisar versiones*
3. *establecer estrategia*
   3.1. *armar versión requerida*
   3.2. *usar una versión específica*
2. *procesar directivas*
   2.1. *especificar servicios*
   2.2. *seleccionar versiones aptas*
   2.3. *acoplar servicios con versiones*
4. *armar estructura de configuración*
5. *enviar mensaje de configuración*

Cuadro 4.85: Detalle de la tarea *Configurar Servicios*

4.5.3. Modelo de Comunicación

El modelo de comunicación de MAS-*Common*KADS *extendido*, nos describe las interacciones entre los agentes involucrados en la resolución de un problema. Este modelo estructura las interacciones en conversaciones. Cada interacción entre dos agentes se realiza mediante el envío de un mensaje, y tiene asociado un acto de habla.

4.5.3.1. Estructura del Modelo de Comunicación

El modelo se estructura en torno a los agentes y las tarea, e introduce los siguientes conceptos [21]:

- *Mensaje:* estructura de datos que intercambian los agentes para comunicarse.
- *Acto de habla:* intención del emisor del mensaje al transmitir el contenido del mismo.
- *Servicio:* prestación realizada por un agente para satisfacer las necesidades de otro agente.
- *Conversación:* conjunto de interacciones cuyo fin es la realización de un servicio.

Basandonos en los casos de uso definidos en la fase de conceptuación, podemos identificar los actos de habla. Los mismos, se pueden distinguir a través de un *diagrama de interacción*

de UML, donde se representan las interacciones entre los agentes del sistema (ver figura 4.2). El resto de la estructura del modelo se complementa con la descripción las conversaciones a través de las plantillas textuales de MAS-*Common*KADS.

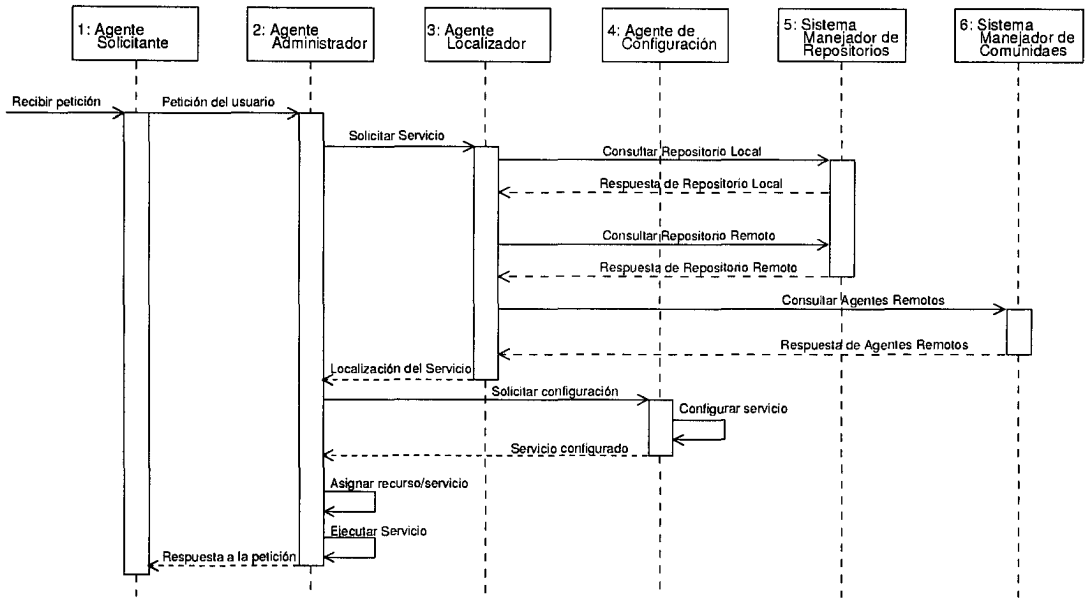

Figura 4.2: Diagrama de interacción de los agentes del sistema

Conversación: *Requerimiento de Usuario*

1

Nombre: Requerimiento de Usuario.
Tipo: Obtención de información.
Objetivo: Satisfacer una petición de usuario.
Agentes: *Solicitante* y *Administrador*.
Iniciador: *Solicitante*.
Servicio: satisfacer-requerimiento.
Actos de habla: *Recibir petición*, *Petición del usuario*, *Respuesta a la petición*.
Descripción: El agente *Solicitante* recibe la información dada por el usuario al realizar un requerimiento al sistema. Esta información es enviada al agente *Administrador*, quien procesa el requerimiento para finalmente enviar un mensaje de respuesta al agente *Solicitante*.
Precondición: Haber recibido un requerimiento del usuario.
Condición de terminación: El agente *Administrador* provee una respuesta al requerimiento del usuario.

Cuadro 4.86: Conversación *Requerimiento de Usuario*

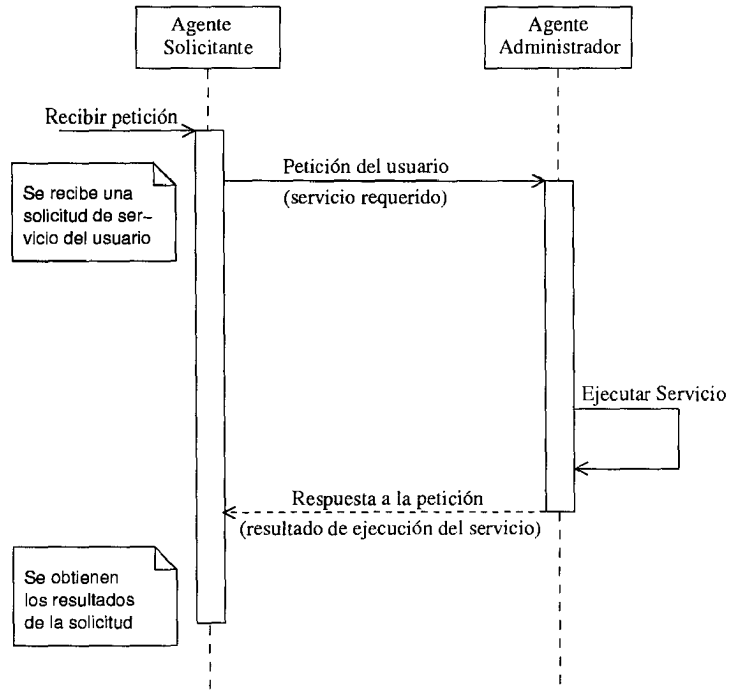

Figura 4.3: Diagrama de Interacción de la conversación *Requerimiento de Usuario*

Conversación: *Localizar Servicio*

2

Nombre: Localizar Servicio.
Tipo: Obtención de información.
Objetivo: Permite la localización de un servicio determinado con el fin de satisfacer una petición de usuario.
Agentes: Sistema Manejador de Repositorios, Sistema Manejador de Comunidades, agente *Localizador*, agente *Administrador*.
Iniciador: *Administrador*.
Servicio: localizar-servicio.
Actos de habla: *Solicitar servicio*, *Localización del servicio*, *Consultar Repositorio Local*, *Respuesta del Repositorio Local*, *Consultar Repositorio Remoto*, *Respuesta del Repositorio Remoto*, *Consultar Agentes Remotos*, *Respuesta de Agentes Remotos*.
Descripción: El agente *Administrador* solicita al agente *Localizador* la ubicación de los servicios y versiones necesarias para satisfacer un requerimiento de usuario. El agente *Localizador* consulta primero al repositorio local del Sistema Manejador de Repositorios, si no se consigue información del servicio buscado, se consulta al repositorio remoto. En caso de no conseguir la información requerida en éste último, se envía la solicitud al Sistema Manejador de Comunidades, quien devuelve un mensaje de respuesta al agente *Localizador*.
Precondición: Haber recibido una solicitud de búsqueda de servicio.
Condición de terminación: El agente *Administrador* recibe como respuesta la localización del servicio requerido.

Cuadro 4.87: Conversación *Localizar Servicio*

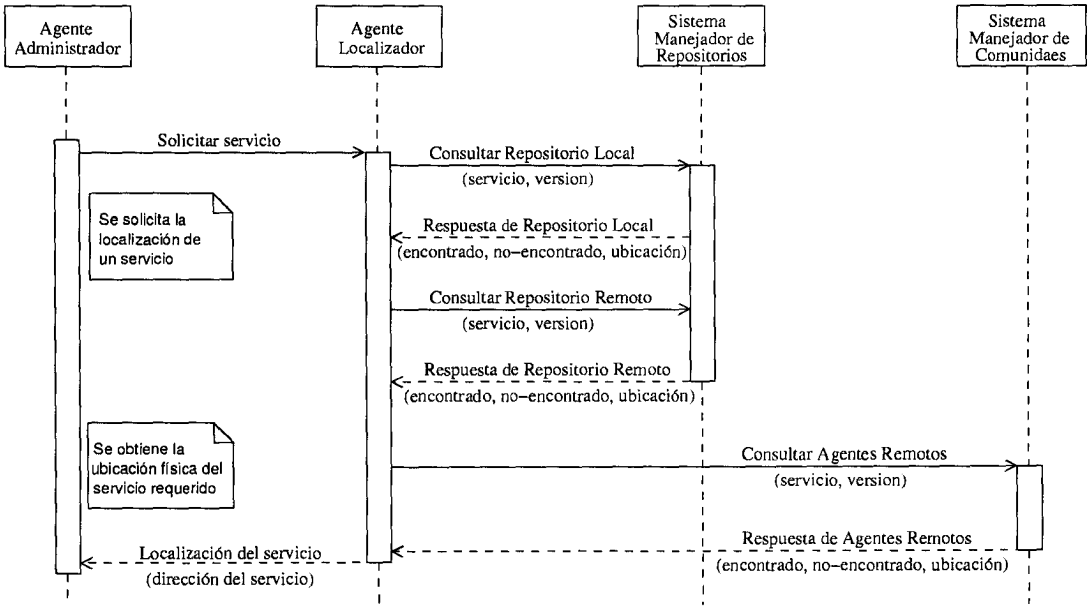

Figura 4.4: Diagrama de Interacción de la conversación *Localizar Servicio*

Conversación: *Configurar Servicio*

3

Nombre: Configurar Servicio.
Tipo: Obtención de información.
Objetivo: Permite la localización de un servicio determinado con el fin de satisfacer una petición de usuario.
Agentes: Agente de *Configuración* y agente *Administrador.*
Iniciador: *Administrador.*
Servicio: configurar-servicio-SOW.
Actos de habla: *Solicitar configuración, Servicio configurado.*
Descripción: El agente *Administrador* envía un mensaje de solicitud al agente de *Configuración* quien realiza las acciones para obtener una adecuada configuración del servicio solicitado. El agente *de Configuración* envía como resultado al agente *Administrador*, el servicio configurado de acuerdo al requerimiento de usuario.
Precondición: Haber finalizado la conversación *Localizar Servicio.*
Condición de terminación: El agente *Administrador* recibe como respuesta el servicio configurado.

Cuadro 4.88: Conversación *Configurar Servicio*

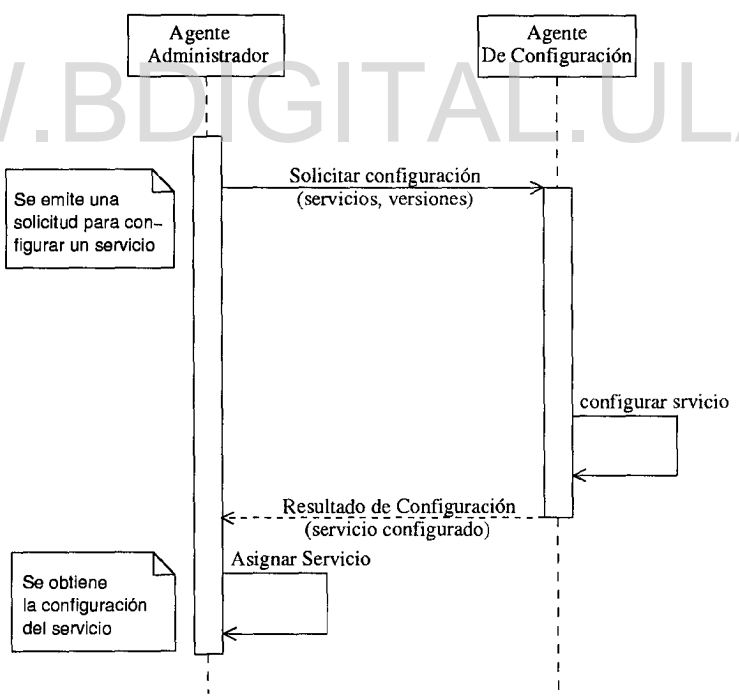

Figura 4.5: Diagrama de Interacción de la conversación *Configurar Servicio*

4.5.4. Modelo de Organización

El modelo de Organización tiene como objetivo analizar desde una perspectiva de grupo las interacciones que hay entre los agentes (tanto de software como humanos) involucrados en el sistema. Así, el modelo describe tanto la organización humana en la que el sistema multiagente va a ser introducido como la organización interna del sistema multiagente. Nuestro Modelo de Organización estará orientado principalmente a modelar las relaciones estáticas entre los agentes del sistema.

Para describir las relaciones estáticas o estructurales de una sociedad multiagente, la metodología MAS-*Common*KADS propone una notación gráfica a través de la cual se representa una jerarquía de clases de agentes. En nuestro Modelo de Organización tenemos claramente identificadas cuatro clases de agentes: *Solicitante*, *Administrador*, *Localizador* y *Configurador*, las cuales constituyen la estructura jerárquica descrita en la figura 4.6.

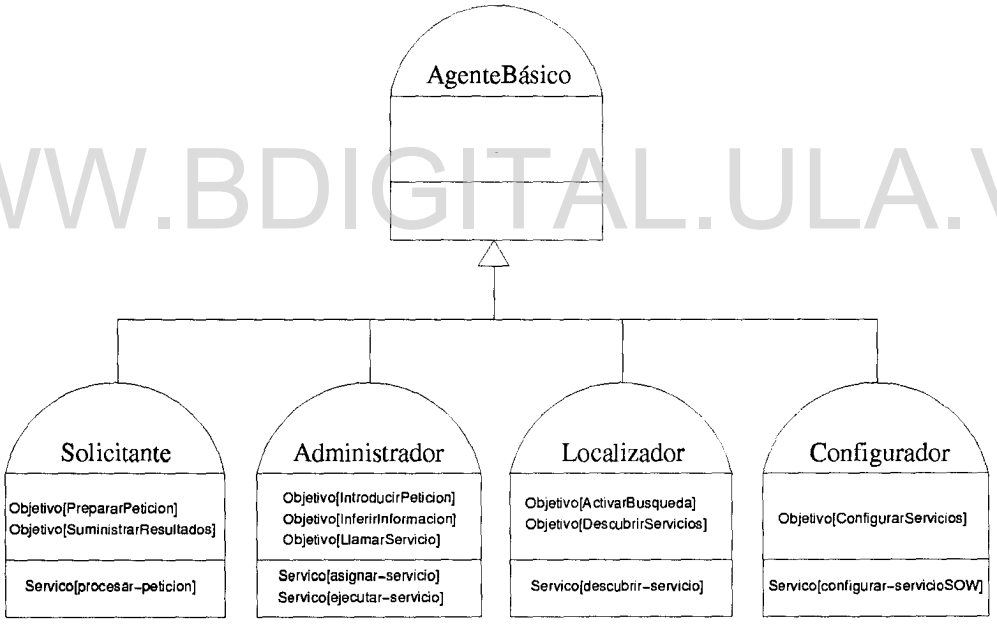

Figura 4.6: Diagrama de clases de agentes para el SMR

4.5.5. Modelo de Inteligencia

En este modelo identificamos las capacidades de razonamiento necesarias para que los agentes del sistema puedan alcanzar sus objetivos. En nuestro sistema hemos identificado dos agentes con capacidades de razonamiento, éstos son, el agente *Administrador* y el agente *Localizador*. A continuación se presenta la estructura del modelo de inteligencia para éstos agentes.

4.5.5.1. Modelo de Inteligencia del Agente Administrador

Mecanismo de Razonamiento 1.1
Fuente de Información: Perfil del usuario. Resultados previos obtenidos del agente de *Configuración* y del agente *Localizador*. Reglas adaptativas. **Fuente de Activación:** Las tareas: *Coordinar Solicitud*, *Caracterizar Versiones*, y *Asignar Recurso/Servicio*. **Tipo de Inferencia:** Basada en reglas. **Relación Tarea-Objetivo:** Decide el método o algoritmo de asignación que se usará en la tarea *Asignar Recurso/Servicio*. **Estrategia de Razonamiento:** Coordinar actividades de búsqueda o configuración a través de estrategias similares (analogía). Enfrentar situaciones desconocidas para enriquecer la experiencia.

Cuadro 4.89: Mecanismo de Razonamiento del Agente Administrador

Mecanismo de Aprendizaje 1.2
Nombre: Retroalimentación **Tipo:** Adaptativo. **Técnica de Representación:** Reglas. **Fuente de Aprendizaje:** Éxito y/o fallas ocurridas durante la coordinación de los procesos de localización y configuración de servicios. **Mecanismo de Actualización:** Las experiencias previas son usadas para actualizar el conocimiento.

Cuadro 4.90: Mecanismo de Aprendizaje del Agente Administrador

Experiencia 1.3

Representación: Reglas
Tipo: Basada en casos.
Grado de Confiabilidad: Medianamente confiable.
Esquema de Procesamiento: Reajuste de parámetros de conocimiento.

Cuadro 4.91: Experiencia del Agente Administrador

Conocimiento Estratégico 1.4

Valor del Conocimiento: Configuración del servicio.
Agente que lo Produce: *Agente de Configuración.*
Proceso que lo Generó: configurar-servicio-SOW.
Grado de Confiabilidad: Altamente confiable.

Cuadro 4.92: Conocimiento Estratégico del Agente Administrador

Conocimiento de Tareas 1.5

Conocimiento de Tareas: 1. Tarea: *Coordinar Búsqueda.*
Agente que la realiza: *Localizador.*

2. Tarea: *Configurar Servicio.*
Agente que la realiza: *Configuración.*

Cuadro 4.93: Conocimiento de Tareas del Agente Administrador

4.5.5.2. Modelo de Inteligencia del Agente Localizador

Mecanismo de Razonamiento	2.1
Fuente de Información: Resultados previos obtenidos de consultas al Sistema Manejador de Repositorios y al Sistema Manejador de Comunidades. Árbol de búsqueda. **Fuente de Activación:** Las tareas: *Coordinar Búsqueda* y *Descubrir Servicio.* **Tipo de Inferencia:** Basada en reglas. **Relación Tarea-Objetivo:** Decide el método o algoritmo de búsqueda que se usará en la tarea *Descubrir Servicio.* **Estrategia de Razonamiento:** Generalizar a partir de la experiencia, la realización de la misma búsqueda con mayor facilidad en el futuro (inducción). Enfrentar situaciones desconocidas para enriquecer la experiencia.	

Cuadro 4.94: Mecanismo de Razonamiento del Agente Localizador

Mecanismo de Aprendizaje	2.2
Nombre: Optimizar búsqueda. **Tipo:** Adaptativo. **Técnica de Representación:** Árbol de búsqueda. Reglas. **Fuente de Aprendizaje:** Resultados de las búsquedas dinámicas, resultados de las consultas realizadas a los sistemas Manejador de Comunidades y Manejador de Repositorios **Mecanismo de Actualización:** Retroalimentación, las experiencias previas son usadas para actualizar el conocimiento.	

Cuadro 4.95: Mecanismo de Aprendizaje del Agente Localizador

Experiencia	2.3
Representación: Reglas **Tipo:** Basada en casos. **Grado de Confiabilidad:** Medianamente confiable. **Esquema de Procesamiento:** Reajuste de parámetros de conocimiento.	

Cuadro 4.96: Experiencia del Agente Localizador

Conocimiento Estratégico	2.4
Valor del Conocimiento: Ubicación del servicio. **Agente que lo Produce:** Sistema de Manejo de Comunidades y Sistema de Manejo de Repositorios. **Grado de Confiabilidad:** Depende de los Sistemas Manejadores de Comunidades y de Repositorios.	

Cuadro 4.97: Conocimiento Estratégico del Agente Localizador

Conocimiento de Tareas	2.5
Conocimiento de Tareas: 1. Tarea: *Asignar Recurso/Servicio.* Agente que la realiza: *Administrador.* 2. Tarea: *Ejecutar Servicio.* Agente que la realiza: *Administrador.*	

Cuadro 4.98: Conocimiento de Tareas del Agente Localizador

4.5.6. Modelo de Coordinación

A través de este modelo representamos las estructuras de comunicación del sistema, además de los protocolos y lenguajes asociados a las comunicaciones. A continuación presentamos la estructura del Modelo de Coordinación del SMR.

4.5.6.1. Modelo de Coordinación de la conversación *Requerimiento de Usuario*

Esquema de Coordinación	1.1
Objetivo a Seguir: Planificar por un lado las interacciones entre los agentes *Solicitante* y *Administrador*, y por otro lado, las interacciones entre el sistema y el usuario. **Tipo:** Predefinido. **Coordinación por Defecto:** Se utilizan mecanismos de pase de mensajes para relacionar los agentes que participan en la conversación. **Actores:** Agente Solicitante, Agente Administrador	

Cuadro 4.99: Esquema de Coordinación de la conversación *Requerimiento de Usuario*

Planificación	1.2
Tipo: Predefinida **Técnica:** Hay un protocolo predefinido entre el el agente *Solicitante* y el agente *Administrador*.	

Cuadro 4.100: Planificación de la conversación *Requerimiento de Usuario*

Mecanismo de Comunicación	1.3
Tipo: Directa. **Técnica empleada:** Pase de mensaje.	

Cuadro 4.101: Mecanismo de Comunicación de la conversación *Requerimiento de Usuario*

Metalenguaje	1.4
Nombre: KQML	

Cuadro 4.102: Metalenguaje de la conversación *Requerimiento de Usuario*

Ontología	1.5
Nombre: Ontología de Interfaz (entradas/salidas). **Representación:** KQML. **Agentes que la conocen:** *Agente Solicitante.* **Vocabulario:** El vocabulario está definido a través clases y relaciones entre clases (ver figura 4.7). **Descripción:** Define el vocabulario manejado por el agente *Solicitante* para realizar actividades de interación entre el sistema y los usuarios.	

Cuadro 4.103: Ontología de Interfaz

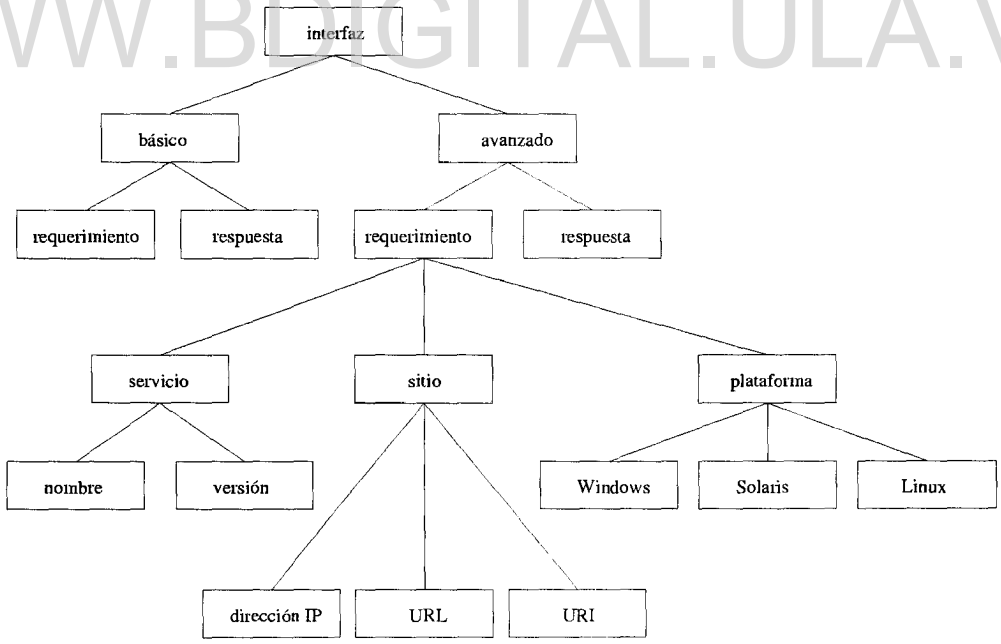

Figura 4.7: Ontología de Interfaz

4.5.6.2. Modelo de Coordinación de la conversación Localizar Servicio

Esquema de Coordinación	2.1
Objetivo a Seguir: Planificar por un lado las interacciones entre los agentes *Localizador* y *Administrador*, y por otro lado, las interacciones entre éste último y los sistemas Manejador de Comunidades y Manejador de Repositorios. usuario. **Tipo:** Predefinido. **Coordinación por Defecto:** Se utilizan mecanismos de pase de mensajes para relacionar los agentes que participan en la conversación *Localizar Servicio*. **Actores:** Agente Localizador, Agente Administrador, Sistema Manejador de Repositorios, Sistema Manejador de Comunidades.	

Cuadro 4.104: Esquema de Coordinación de la conversación *Localizar Servicio*

Planificación	2.2
Tipo: Demanda-Oferta y Jerarquía. **Técnica:** Los agentes participantes siguen un modelo de interacciones guiados por los resultados parciales que se van encontrando durante el proceso de búsqueda, siguiendo una jerarquía que pasa primero por el SMR y después por el Sistema Manejador de Comunidades	

Cuadro 4.105: Planificación de la conversación *Localizar Servicio*

Mecanismo de Comunicación	2.3
Tipo: Directa. **Técnica empleada:** Pase de mensaje.	

Cuadro 4.106: Mecanismo de Comunicación de la conversación *Localizar Servicio*

Metalenguaje	2.4
Nombre: KQML	

Cuadro 4.107: Metalenguaje de la conversación *Localizar Servicio*

Ontología	2.5
Nombre: Ontología de Comunicación SOW. **Representación:** KQML. **Agentes que la conocen:** *Agente Localizador*, Sistema Manejador de Repositorios y Sistema Manejador de Comunidades.. **Vocabulario:** El vocabulario está definido a través clases y relaciones entre clases (ver figura 4.8). **Descripción:** Es una ontología global del SOW, a través de la cual se define el vocabulario que es manejado en las comunicaciones que se establecen entre los distintos subsistemas del SOW.	

Cuadro 4.108: Ontología de Comunicación SOW

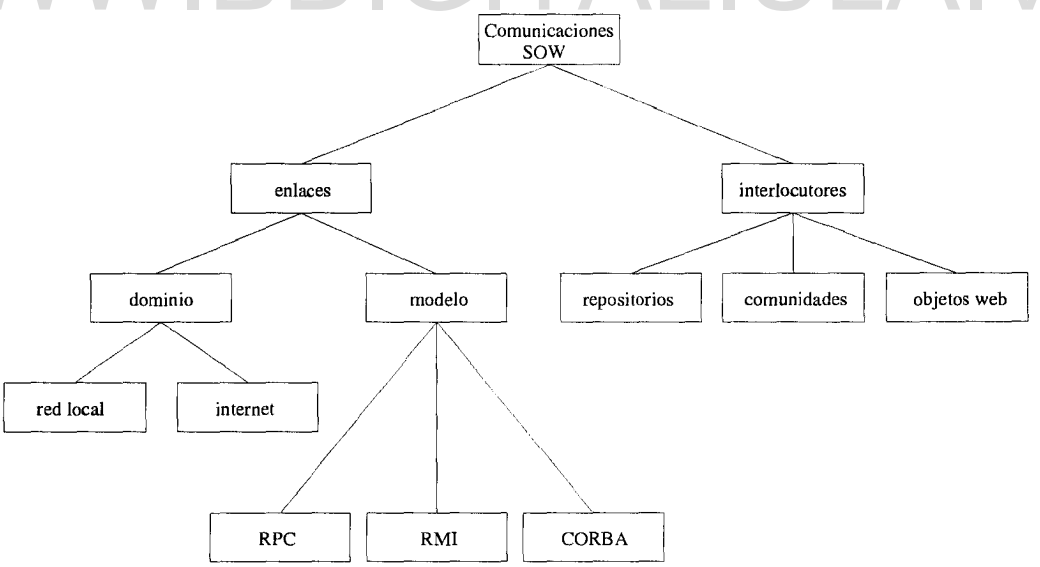

Figura 4.8: Ontología de Comunicación SOW

4.5.6.3. Modelo de Coordinación de la conversación *Configurar Servicio*

Esquema de Coordinación	3.1
Objetivo a Seguir: Coordinar las acciones efectuadas por los agentes Administrador y de Configuración para llevar a cabo la configuración y asignación de servicios **Tipo:** Predefinido. **Coordinación por Defecto:** Se utilizan mecanismos de pase de mensajes para relacionar los agentes que participan en la conversación *Configurar Servicio.* **Actores:** Agente Administrador, Agente de Configuración.	

Cuadro 4.109: Esquema de Coordinación de la conversación *Configurar Servicio*

Planificación	3.2
Tipo: Centralizada **Técnica:** Las actividades de coordinación de esta conversación se centran en el agente *Administrador.*	

Cuadro 4.110: Planificación de la conversación *Configurar Servicio*

Mecanismo de Comunicación	3.3
Tipo: Directa. **Técnica empleada:** Pase de mensaje.	

Cuadro 4.111: Mecanismo de Comunicación de la conversación *Configurar Servicio*

Metalenguaje	3.4
Nombre: KQML	

Cuadro 4.112: Metalenguaje de la conversación *Configurar Servicio*

Ontología 3.5

Nombre: Ontología de Servicios SMR.
Representación: KQML.
Agentes que la conocen: *Agente Solicitante, Agente Administrador, Agente Localizador*, y *Agente de Configuración.*
Vocabulario: El vocabulario está definido a través clases y relaciones entre clases (ver figura 4.9).
Descripción: Esta ontología describe las entidades, conceptos y relaciones empleadas para satisfacer las peticiones de servicio en el SMR.

Cuadro 4.113: Ontología de Servicios SMR

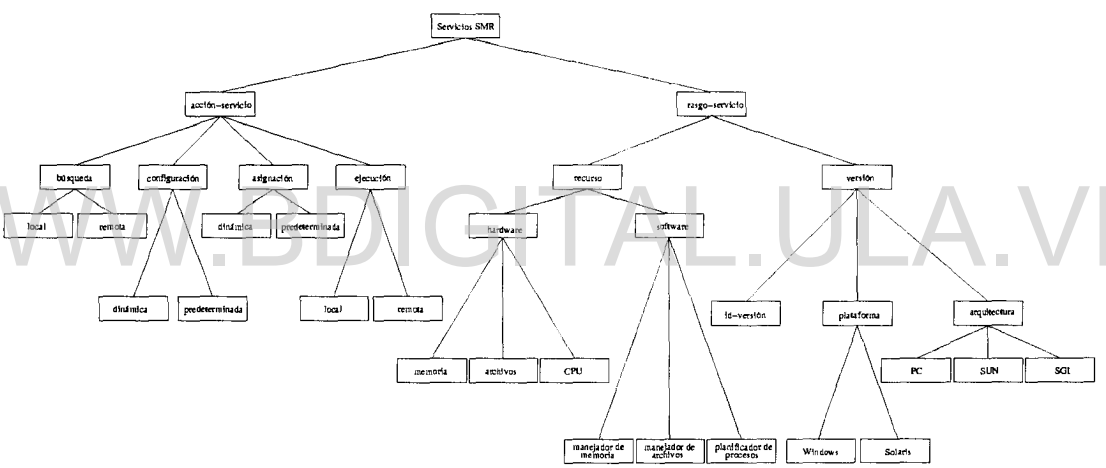

Figura 4.9: Ontología de Servicios SMR

Capítulo 5

Implementación y prueba de un prototipo de SMR

5.1. Introducción

El presente capítulo describe la parte experimental de este trabajo. En él cubriremos, por un lado, aspectos relativos a la implementación del prototipo del SMR, y por otro lado, a la evaluación del prototipo en situaciones que permitan medir ciertos rasgos del sistema, con el fin de estudiar su desempeño.

Particularmente, para el análisis de nuestra propuesta se realizan simulaciones; la simulación es una vía factible para lograr de forma rápida y económica el análisis y la prueba de las partes que conforman el sistema distribuido de administración de recursos heterogéneos.

5.2. Simulación del SMR

El modelo de SMR que simulamos está basado en el diseño desarrollado en el capítulo 4. La simulación trabaja sobre la misma arquitectura de software descrita en dicho capítulo, es decir, presentamos una implementación orientada a Sistemas Multiagente donde se simulan las actividades de cada uno de los agentes del SMR y sus interrelaciones.

Las actividades fundamentales del SMR son: la búsqueda, la configuración y la asignación de recursos. La búsqueda de recursos es una actividad que el SMR lleva acabo en coordinación con otros subsistemas del SOW, como lo son el Sistema Manejador de Comunidades y el Sistema Manejador de Repositorios. Por otro lado, la configuración y la asignación de

recursos son actividades exclusivas del SMR. Dentro del contexto de nuestro sistema, éstas actividades son realizadas por los agentes correspondientes mediante algoritmos que se apoyan en estratégias dinámicas y adaptativas, explotando el conocimiento manejado por los agentes. Particularmente, presentamos un algoritmo de configuración y otro de asignación de recursos, los cuales son usados por el agente de *configuración* y el agente *administrador*, respectivamente. Los algoritmos presentados se adaptan perfectamente al modelo diseñado, ya que son algoritmos distribuidos, los cuales trabajan en un ambiente dinámico donde el conocimiento manejado por los agentes es esencial. Una ventaja de nuestro sistema, es la posibilidad de incorporar nuevas estrategias de búsqueda, configuración y asignación de recursos sin mayor problema.

5.2.1. Algoritmo de Configuración

Usamos un algoritmo para la configuración de servicios, el cual está basado en las ideas planteadas en [16]. El algoritmo se fundamenta en el manejo de demandas de componentes y versiones de sistemas. El proceso de configuración es iniciado por un requerimiento específico, a través de este proceso se satisface el requerimiento construyendo un sistema con los componentes y versiones disponibles.

El algoritmo maneja una base de conocimiento, la cual comprende reglas que están estructuradas de manera tal que cada version de servicio tiene asociado una serie de componentes y sus respectivas versiones. Éstos componentes, a su vez, tienen asociados una serie de subcomponentes con sus respectivas versiones, y asi sucesivamente, hasta llegar a los componentes atómicos.

El algoritmo comienza con una solicitud de servicio *probl* y una configuración vacía $conf$, y va descomponiendo paso a paso el problema *probl*, a medida que extiende la configuración $conf$, hasta que en el enésimo paso se obtiene un problema $probl_n$ con componentes atómicos y una configuración $conf_n$ que llega a ser la solución al problema original *probl*. A continuación una descripción formal del modelo en el cual se basa el algoritmo.

Para $n, m \in \mathbb{N}^+$ denotamos a un conjunto de versiones como

$$VER = \{v_1, \ldots, v_m\}$$

Y el conjunto de componentes como

$$COMP = \{c_1, \ldots, c_n\}$$

La relación entre ambos conjuntos está descrita por

$$f_i = \{v_{i,1}, \ldots, v_{i,j}\}$$

f_i representa el conjunto de versiones del componente c_i, es decir que $v_{i,j}$ es la j-ésima versión del componente c_i.

Una configuración viene dada por:

$$conf = \{(c_1, v_{1,k}), \ldots, (c_i, v_{i,j})\}$$

donde $1 \leq i \leq n$ y $1 \leq k, j \leq m$

Una solicitud de servicio es definida por

$$probl = \{servicio_1, \ldots, servicio_m\}$$

Ahora podemos ver que el problema de configuración a solucionar por el algoritmo es definido de la siguiente manera:

Instancia: Dados los conjuntos

$COMP = \{c_1, \ldots, c_n\}$, $VER = \{v_1, \ldots, v_m\}$ por cada elemento de COMP, ambos relacionados por $f_i = \{v_{i,1}, \ldots, v_{i,j}\}$ y dada una petición de servicio $probl = \{servicio_1, \ldots, servicio_m\}$

Consulta: ¿Existe una configuración $conf = \{(c_1, v_{1,k}), \ldots, (c_i, v_{i,k})\}$ tal que $conf$ sea una solución a *probl*?

En la figura 5.1 se presenta el algoritmo en cuestión, podemos ver que es un algoritmo que descompone los servicios en sus respectivos componentes atómicos, con sus respectivas versiones, según lo definido en la base de conocimiento. El algoritmo se ejecuta mientras existan peticiones en la lista *probl* (línea 01). La función *descomp* (línea 02) se encarga de

descomponer recursivamente cada uno de los servicios a configurar, cada servicio va siendo descompuesto recursivamente (líneas 03 y 04), siempre y cuando el componente no sea atómico (línea 05). En cada iteración se extiende la lista *conf*, al agregar la versión del componente (línea 09). Finalmente, por cada servicio configurado se reduce la lista *probl* (línea 13).

```
01: while ∃ (servicio) ∈ probl do
02:     function descomp(servicio)
03:         n = num_comp(servicio)
04:         for i = 1 to n do
05:           if comp(c_i) ≠ atomico
06:             descomp(c_i)
07:             else
08:               buscar_version(v_{i,k})
                   compatible con conf
09:               conf = {conf,(c_i, v_{i,k})}
10:           endif
11:         endfor
12:     endfunction
13:     probl = probl - {servicio}
14: endwhile
```


Figura 5.1: Algoritmo de Configuración

5.2.2. Algoritmo de Asignación

Presentamos un algoritmo de asignación de servicios/recursos para sistemas multiagentes. El algoritmo propuesto permite la asignación en línea de recursos o servicios distribuidos. A continuación una descripción del algoritmo.

Consideramos un conjunto de servicios que deben ser ejecutados (ya los servicios han sido descompuestos en sus componentes atómicos, por lo cual se tienen definidas la versiones a usar), con sus repectivas configuraciones definidas en el conjunto de configuraciones

$$conf = \{(c_1, v_{1,k}), \ldots, (c_i, v_{i,k})\}$$

y un conjunto de máquinas disponibles para ejecutar dichos servicios

$$M = \{m_1, ..., m_n\}$$

En cada $m_j \in M$ implicitamente se hayan recursos computacionales como memoria, tiempo de CPU, etc. Los servicios asignados a las máquinas de M hacen uso de estos recursos.

El problema de asignación a solucionar por el algoritmo es definido de la siguiente manera:

Instancia: Dados los conjuntos

$$conf = \{(c_1, v_{1,k}), \ldots, (c_i, v_{i,k})\} \text{ y } M = \{m_1, ..., m_n\}$$

Consulta: ¿Existe una asignación tal que para todo c_i en $conf$, existe un $m_j \in M$?

El algoritmo busca asignar una configuración de servicio c_i a una máquina m_j que se encuentre disponible para ejecutar el servicio. El algoritmo maneja una base de conocimiento, la cual comprende reglas que están estructuradas de manera tal que cada servicio configurado tiene asociado una lista de máquinas disponibles para ejecutar el servicio.

```
01: while ∃ c_i ∈ conf
02:     sacar_de_lista(conf, c_i)
03:     if ∃ m_j ∈ M and m_j es maquina_libre
        and ∃(c_i, v_i,k) en m_j then
04:       asignar(c_i, m_j)
05:     else
06:       fallo_asignacion
07:     endif
08: endwhile
```

Figura 5.2: Algoritmo de Asignación

5.2.3. Búsqueda y descubrimiento de servicios

Como se mencionó antes, las actividades de búsqueda de servicio son coordinadas por los diferentes subsistemas del SOW (subsistema manejador de recursos, subsistema manejador de repositorios, subsistema manejador de objetos web y subsistema manejador de comunidades). A través de la interacción entre diversos agentes, se definen las actividades de búsqueda

y descubrimiento de servicios en el SOW, las cuales se realizan de forma dinámica, siendo fundamental para éstas actividades el uso de motores de búsqueda en interacción con los repositorios y las comunidades de nodos del SOW. Tanto los repositorios como las comunidades son manejados por agentes externos al SMR, en consecuencia, un algoritmo de búsqueda para el SOW involucra la interacción de agentes del SMR con agentes de otros subsistemas del SOW. Es por esto que en el presente trabajo, el cual está circunscrito al SMR del SOW, no hemos considerado un algoritmo de búsqueda en la simulación que presentamos, sin embargo, exponemos algunos aspectos relacionados con la búsqueda y descubrimiento de servicios en el SOW. Solo se simulan las llegadas de las respuestas a las solicitudes de búsqueda generadas por otros subsistemas del SOW, ya que los resultados de las solicitudes de búsqueda son usados por el SMR.

El SOW provee una infraestructura a través de la cual se produce una eficiente comunicación entre agentes, lo cual permite aplicar métodos y estrategias que facilitan las labores de búsqueda en el SOW, tales como: manejo de comunidades de nodos, migración y réplica de objetos móviles, y manejo de cache en la Web. Bajo éste enfoque, la búsqueda en el SOW se orienta a un paradigma de búsqueda *de Agente a Agente* (A2A) [31], donde las búsquedas locales y globales están determinadas por la información que intercambian los agentes, la cual es producto del conocimiento manejado por cada agente y el empleo de mecanismos inteligentes para la la interpretación de dicho conocimiento. En un trabajo fúturo se integrarán los agentes de los distintos subsistemas del SOW para estructurar un método de búsqueda A2A que permita expandir dinámicamente la búsqueda de servicios a través de la web.

En el SOW propuesto se presentan tres niveles de búsqueda coordinadas por el SMR para satisfacer una solicitud de servicio dada.

1. *Nivel Local*: en primer lugar, se busca en los repositorios locales a través del Sistema Manejador de Repositorios Locales. En caso de tener éxito, se ofrece una respuesta inmediata, en caso contrario, se pasa al siguiente nivel de búsqueda .

2. *Nivel Remoto*: se busca en los repositorios remotos a través del Sistema Manejador de Repositorios Remotos, esto es, consultar a los nodos vecinos. Si no hay éxito en la búsqueda, se pasa al siguiente nivel.

3. *Nivel de Comunidades*: se realiza la búsqueda en las comunidades existentes a través del

Sistema Manejador de Comunidades. En caso de no tener éxito, no puede ser satisfecha la solicitud.

5.3. Análisis de desempeño

En esta sección presentamos resultados prácticos obtenidos producto del proceso de experimentación efectuado con el modelo de simulación del SMR. Las simulaciones empleadas en este trabajo han sido desarrolladas a través de una herramienta para desarrollo de simulaciones basadas en agentes, conocida como SeSAm [25].

El modelo de simulación desarrollado realiza simulaciones de los procesos de búsqueda, configuración y asignación de recursos. El análisis de desempeño presentado en esta sección está basado en estos tres procesos, sobre los cuales se aplican los siguientes criterios de rendimiento:

1. *Eficiencia:* Se refiere a la capacidad que posee el sistema para atender las peticiones a la mayor brevedad posible. La eficiencia es interpretada como una expresión de la velocidad de respuesta del sistema. Para observar como influye la eficiencia en el desempeño del sistema usamos tres métricas, a las que hemos llamado *ea*, *ec*, *eb*, las cuales serán usada en las secciones 5.3.1, 5.3.2 y 5.3.3 para medir el desempeño de los procedimientos de asignación, configuración y búsqueda de recursos, respectivamente. En general, el valor promedio de la eficiencia viene dado por la siguiente fórmula:

$$e = \frac{np}{nc} \tag{5.1}$$

donde np es el número de peticiones procesadas en un lapso de tiempo dado y nc es el número de consultas realizadas en dicho lapso.

2. *Inteligencia:* Se refiere a la habilidad que poseen los agentes del sistema para prestar los servicios de forma confiable, haciendo uso de técnicas de razonamiento. Para el análisis de desempeño que presentamos, hemos considerado la *inteligencia* como un criterio de rendimiento valioso, ya que en el SMR se propone esencialmente el uso de

técnicas inteligentes para el manejo de recursos, y se hace necesario medir el impacto que produce el uso de estas técnicas en los procedimientos de asignación y configuración de servicios/recursos. Hemos definido dos parámetro a través de los cuales se puede expresar la manera en que la utilización de técnicas inteligentes afecta el desempeño de dichos procedimientos. Uno de estos parámetros lo hemos denominado *ia* y es usado en la sección 5.3.1 para indicar la probabilidad de realizar una asignación exitosa en el primer intento, un segundo parámetro lo hemos denominado *ic* y es usado en la sección 5.3.2 para indicar el grado de conocimiento que el *agente de configuración* posee con respecto a las configuraciones que son requeridas durante el proceso de configuración. Para analizar el proceso de búsqueda no se ha usado el criterio de inteligencia, ya que algunos actores que manejan técnicas de inteligencia en el proceso de búsqueda no forman parte del SMR.

5.3.1. Análisis de desempeño para el proceso de asignación

Para que el *agente administrador* pueda designar los nodos de la red que serán usados para ejecutar ciertos recursos/servicios solicitados, es necesario realizar cierto número de consultas a los nodos disponibles para determinar cual es el nodo que va a proveer el recurso/servicio requerido, esto implica cierto número de comunicaciones. El desempeño del algoritmo de asignación planteado en la sección anterior depende del número de consultas realizadas, para un menor número de consultas tenemos un proceso de asignación rápido, y un uso eficiente de recursos tales como el tiempo de servicio y el ancho de banda, entre otros.

Establecemos la *Eficiencia de la Asignación* como una métrica importante para el análisis de desempeño en el proceso de asignación del SMR. La eficiencia para el proceso de asignación viene dada por:

$$ea = \frac{r}{ca} \tag{5.2}$$

donde r es el número de total de recursos/servicios a ser asignadas y ca es el número total de consultas realizadas durante el proceso de asignación. Hemos realizado cierto número de experimentos para precisar los valores de las métricas ca y ea asignando al parámetro r el siguiente rango de valores

r: 2000, 4000, 6000, 8000, 10000, 12000

estos valores son representativos de cantidades de peticiones que eventualmente pueden requerir ser asignadas por el SMR. Para estos valores de r hemos obtenido los valores de ca que se muestran en la tabla 5.1, los valores de ea fueron calculados a través de la fórmula 5.2. Hemos realizado los experimentos del proceso de asignación suponiendo que se requiere más de un intento para conseguir el sitio donde seran asignados los recursos/servicios (más adelante veremos que sucede cuando se puede realizar la asignación en el primer intento)

El comportamiento descrito por los valores presentados en la tabla 5.1 se expresa en las figuras 5.3, 5.4 y 5.5

r	2000	4000	6000	8000	10000	12000
ca	2163	4791	7707	11339	14292	17441
ea	0.92	0.83	0.78	0.71	0.70	0.69

Cuadro 5.1: Experimentos para el proceso de asignación

En la figura 5.3 se aprecia como el número de consultas se incrementa rápidamente a medida que aumenta el número de requerimientos, vemos además que el número de consultas

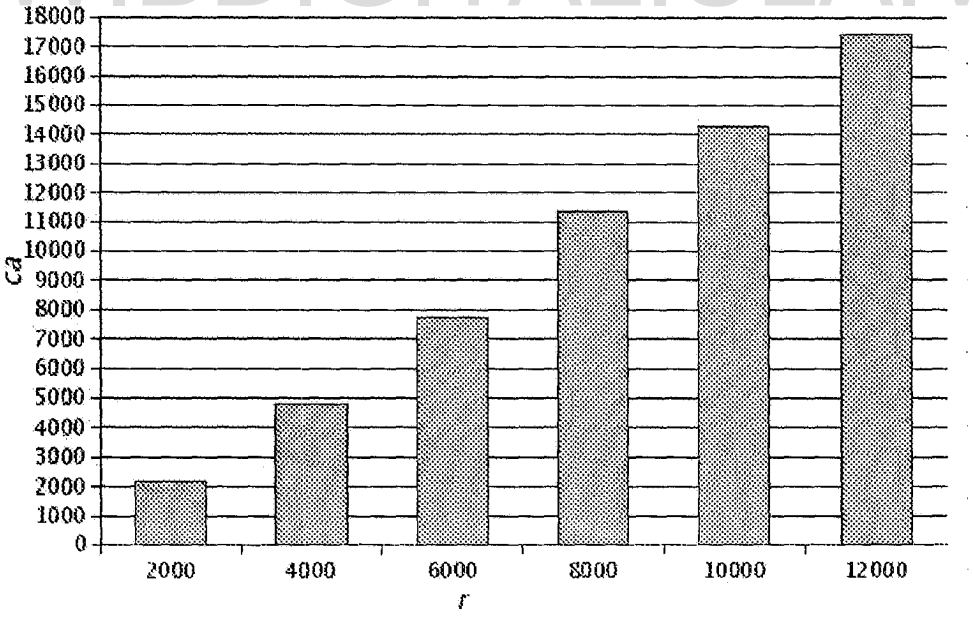

Figura 5.3: Número de consultas en relación a Número de requerimientos

influye sobre la eficiencia en la asignación, esto se aprecia en la figura 5.4, donde se manifiesta una tendencia a disminuir la eficiencia de la asignación a medida que aumenta el número de consultas. Este comportamiento es esperado ya que en la fórmula 5.2 podemos ver que el número de consultas *ca* es inversamente proporcional a la eficiencia de la asignación *ea*; pero por otro lado, el número de requerimientos r es proporcional a la eficiencia de la asignación, sin embargo en la figura 5.5 se manifiesta un comportamiento contrario a lo esperado, es decir, se manifiesta una tendencia a disminuir la eficiencia en la asignación *ea* a medida que aumenta el número de requerimientos r, este comportamiento obedece a los siguientes hechos:

1. *ca* crece rápidamente en relación a r (fig. 5.3)
2. *ea* disminuye en la medida que crece *ca* (fig 5.4)

Así, vemos en la fórmula 5.2 que *ca* crece mas rápido que r lo que provoca que *ea* tiende a disminuir, esta no es una situación beneficiosa para el rendimiento del sistema, así tenemos que el proceso de asignación del SMR no es escalable en relación al número de peticiones.

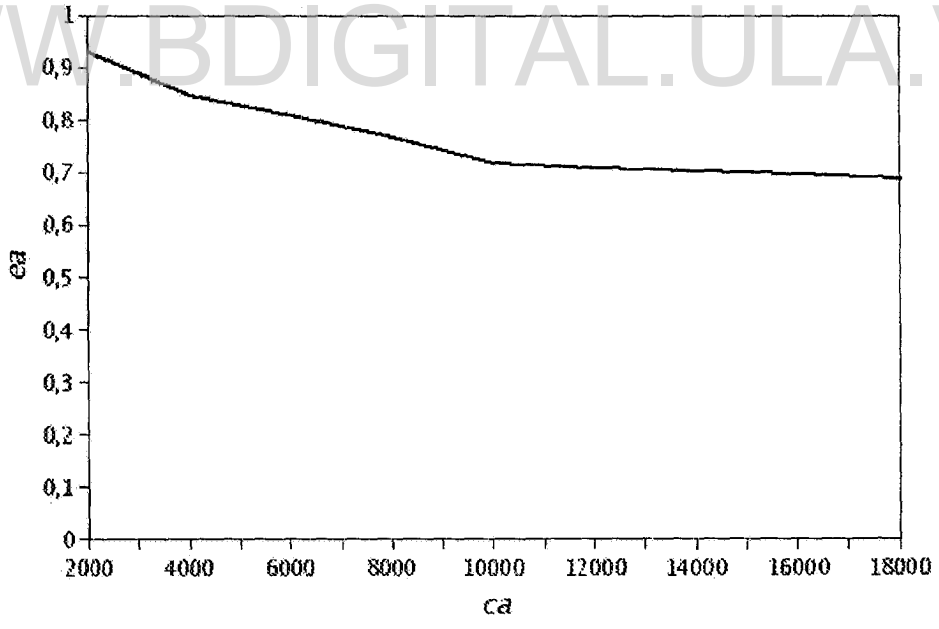

Figura 5.4: Eficiencia de asignación en relación a número de consultas

En la simulación del proceso de asignación se determina de forma aleatoria si una máquina está disponible para realizar la asignación, cuando una máquina no está disponible se intenta encontrar otra máquina disponible. Una situación deseable es que se encuentre una máquina disponible en el primer intento, para alcanzar esa situación, el SMR se sirve del conocimiento manejado por el *agente administrador* haciendo uso de técnicas de inteligencia. Consideramos ahora el criterio de *inteligencia* en el presente estudio, para ello definimos un parámetro *ia*, el cual expresa el grado de cooperación que provee un método de inteligencia dentro del proceso de asignación de servicios/recursos. El parámetro *ia* determina la probabilidad de éxito para encontrar una máquina libre en el primer intento durante el proceso de asignación, valiendose del uso de técnicas de inteligencia. En los experimentos realizados *ia* toma el siguiente dominio de valores:

ia: 0.0; 0.05; 0.10; 0.25; 0.50; 0.80; 1.0

Por ejemplo, si *ia* vale 0.05 la probabilidad de realizar la asignación en el primer intento es de 5 %, cuando *ia* vale 0.10 dicha probabilidad es de 10 %.

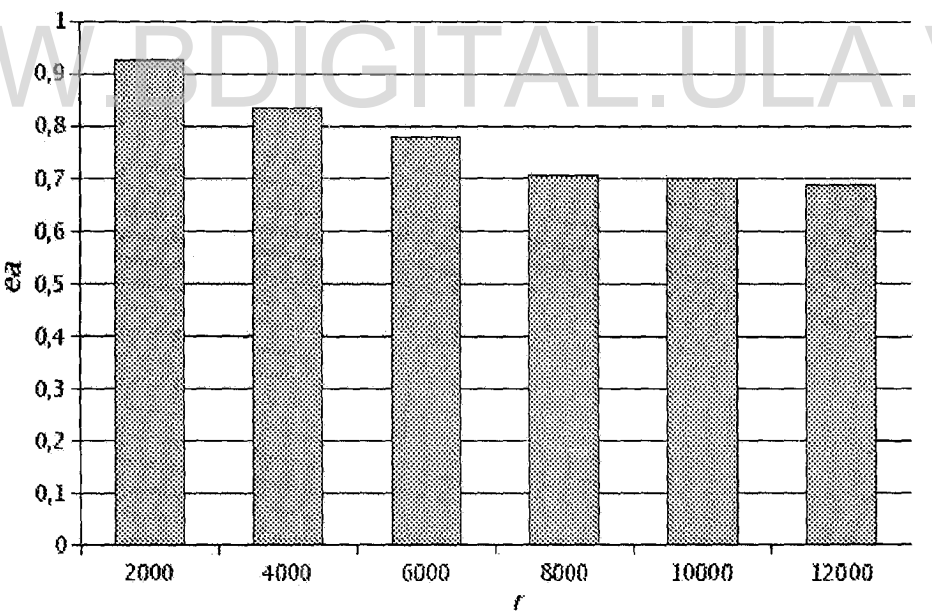

Figura 5.5: Eficiencia de asignación en relación a número de requerimientos

Realizamos seis experimentos con los distintos valores definidos en el rango de ia. Para dichos experimentos fijamos el número de requerimientos r' en 8000 como caso estándar, ya que nos interesa observar el comportamiento de la eficiencia de la asignación ea' y el número de consultas en la asignación ca' sobre una misma instancia de r' a medida que varía el parámetro ia. Los resultados obtenidos son mostrados en la tabla 5.2. El número de consultas ca' se obtiene al sumar el número de consultas requeridas para hacer las asignaciones en el primer intento (lo llamamos c_1) con el número de consultas requeridas para hacer las asignaciones en más de un intento (lo llamamos c_2). Tenemos entonces que

$$c_1 = r' = 8000$$
$$c_2 = (1 - ia) * ca_{8000}$$
$$ca' = c_1 + c_2$$

donde ca_{8000} es el valor de ca en la tabla 5.1 cuando r vale 8000, en otras palabras, ca_{8000} representa la cantidad de consultas realizadas cuando se necesita más de un intento para realizar la asignación de 8000 requerimientos. Los valores de ea se obtienen de la fórmula 5.2.

	ia						
	0.0	**0.05**	**0.10**	**0.25**	**0.50**	**0.80**	**1.00**
r'	8000	8000	8000	8000	8000	8000	8000
ca'	19339	18772	18205	16504	13669	10267	8000
ea'	0.41	0.43	0.44	0.48	0.59	0.78	1.00

Cuadro 5.2: Experimentos para distintos valores de ia

Vimos anteriormente que el número de consultas (ca) afecta sensiblemente a la eficiencia de la asignación (ea), por lo tanto se debe aspirar a reducir el número de consultas, esto pudiera lograrse si se aplican técnicas de inteligencia que permitan al *agente administrador* hacer uso de información almacenada para realizar asignaciones de recursos/servicios con el menor número de consultas posible. Un reflejo de esta situación se expresa través del parámetro ia. El resultado de variar este parámetro se refleja en la figura 5.6 donde se puede ver como se logra disminuir el número de consultas (ca) al aumentar el valor del parámetro ia. De experimentos anteriores sabemos que reducir el número de consultas (ca) implica un aumento en la eficiencia de la asignación (ea), así a medida que aumenta el valor del parámetro ia se

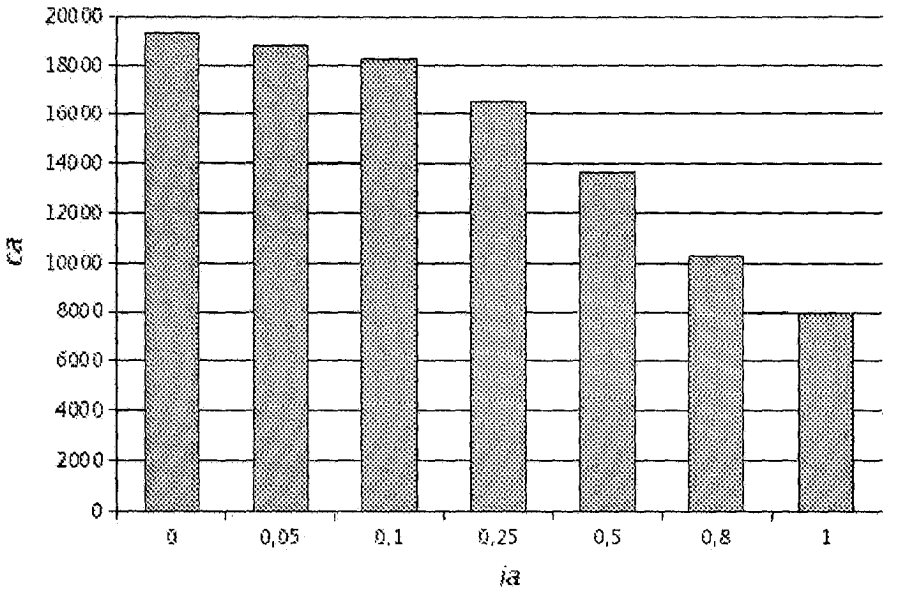

Figura 5.6: Número de consultas en relación a *ia*

consigue aumentar la eficiencia de la asignación *ea*, tal como se muestra en la figura 5.7. Vemos entonces que la cooperación que puede proveer un método de inteligencia dentro del proceso de asignación de servicios/recursos ayuda a mejorar el rendimiento de dicho proceso.

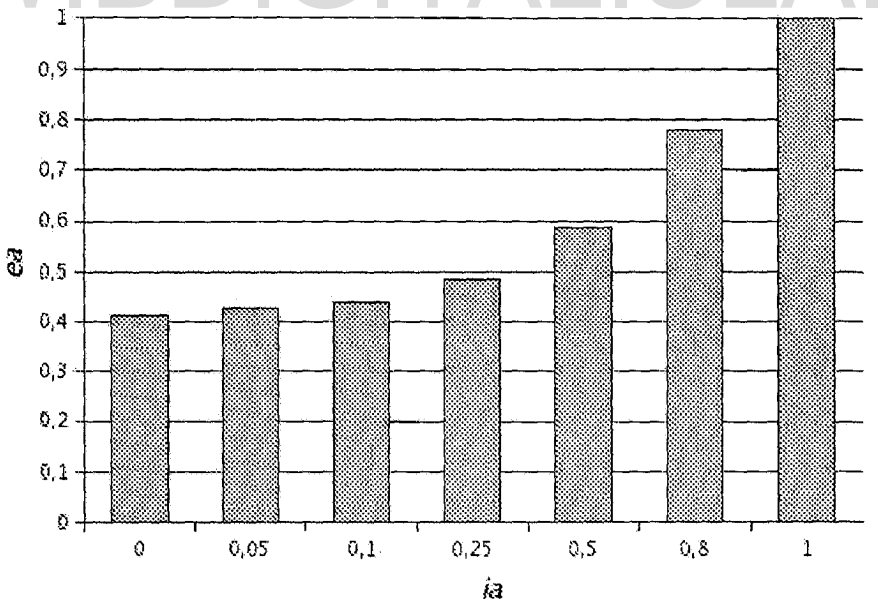

Figura 5.7: Eficiencia de asignación en relación a *ia*

Otra métrica que hemos considerado importante para el estudio del proceso de asignación del SMR es la *utilización*, esta métrica la definimos como el período de tiempo en el que el *agente administrador* utiliza recursos del sistema para resolver el problema de asignación. El criterio de *utilización* está relacionado con el tiempo de servicio y el ancho de banda.

- El *tiempo de servicio* (ts) es el tiempo empleado en servir una petición para asignación de servicios/recursos. Es expresado como un cociente entre unidades de tiempo y número de peticiones.
- El *ancho de banda* (ab) corresponde al número de peticiones que pueden ser atendidas en la red en una unidad de tiempo. Es expresado como un cociente entre número de peticiones y unidades de tiempo.

La *utilización* viene dada por

$$utilizacion = t_{cpu} + t_{ab} \tag{5.3}$$

donde t_{cpu} y t_{ab} vienen dados por:

$$t_{cpu} = r * ts \tag{5.4}$$

$$t_{ab} = \frac{r}{ab} \tag{5.5}$$

Realizamos experimentos para observar el comportamiento de las métricas antes descritas, los experimentos se realizaron en base a los siguientes rangos de valores:

r : $2000, 4000, 6000, 8000, 10000, 12000$

ab : $10 \ldots 60$

ts : $0{,}01 \ldots 0{,}50$

En la simulación, el tiempo lo estamos midiendo en función de la cantidad de consultas empleadas para realizar las asignaciones. Se han realizado experimentos para medir la *utilización* en el SMR, obteniendo los resultados que se muestran en la tabla 5.3.

r	2000	4000	6000	8000	10000	12000
ab	26	10	32	11	6	8
ts	0.28	0.24	0.13	0.1	0.08	0.11
t_{cpu}	560	960	780	800	800	1320
t_{ab}	77	400	188	727	1667	1500
utilizacion	637	1360	968	1527	2467	2820

Cuadro 5.3: Valores de *utilización* en el proceso asignación

Durante la simulación se generan valores aleatorios para el ancho de banda (ab) y el tiempo de servicio (ts). Los valores de t_{cpu} y t_{ab} son obtenidos a través de las fórmulas 5.4 y 5.5 respectivamente. En la figura 5.8 se aprecia que la *utilización* aumenta a medida que aumenta el número de requerimientos, este es un resultado esperado, ya que en las fórmulas 5.4 y 5.5 se ve claramente que r es directamente proporcional a t_{cpu} y t_{ab}, y por lo tanto a la *utilización*. Aunque la *utilización* describe un comportamiento regular en la gráfica, no sucede igual con t_{ab} y t_{cpu} . Este comportamiento irregular de t_{ab} y t_{cpu} obedece a la asignación aleatoria de valores a ab y ts dentro del rango establecido, pero sin seguir ningún patrón determinado, esto es así ya que hemos tomado como referencia la apreciación de [28] donde se plantea que en una estructura compleja como la web no se puede precisar *a priori* los tiempos de servicio así como el ancho de banda disponible.

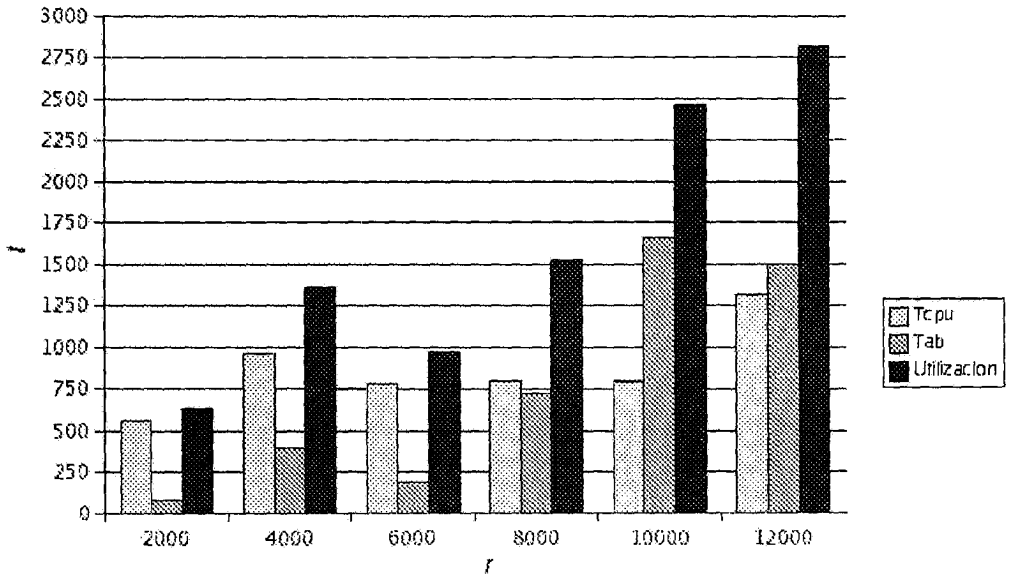

Figura 5.8: tiempo de servicio en relación a utilización

5.3.2. Análisis de desempeño para el proceso de configuración

El desempeño del proceso de configuración depende del número de consultas realizadas entre el *agente administrador* y el *agente configurador*. Establecemos la *Eficiencia de Configuración* como una métrica importante para el análisis de desempeño en el proceso de configuración del SMR. La eficiencia para el proceso de configuración viene dada por:

$$ec = \frac{r}{cc} \tag{5.6}$$

donde r es el número de total de recursos/servicios a ser asignadas y cc es el número total de consultas realizadas durante el proceso de configuración. Hemos realizado cierto número de experimentos para precisar los valores de las métricas cc y ec asignando al parámetro r el siguiente rango de valores

r: 2000, 4000, 6000, 8000, 10000, 12000

Para estos valores de r hemos obtenido los valores de cc que se muestran en la tabla 5.4, los valores de ec fueron calculados a través de la fórmula 5.6. Hemos realizado los experimentos suponiendo que todas las configuraciones requeridas son desconocidas para el *agente de configuración* (más adelante vermos que sucede cuando algunas configuraciones son conocidas por el *agente de configuración*).

r	2000	4000	6000	8000	10000	12000
cc	2103	4315	6282	8745	11087	13526
ec	0.95	0.93	0.96	0.91	0.90	0.89

Cuadro 5.4: Experimentos para el proceso de configuración

En la tabla 5.4 se aprecia como el número de consultas se incrementa regularmente a medida que aumenta el número de requerimientos, como consecuencia de esto, se manifiesta una tendencia a mantener estable en la eficiencia en la asignación ea a medida que aumenta el número de requerimientos. Vemos entonces que el número de consultas cc no sufre un incremento muy fuerte al aumentar el número de requerimientos r, tenemos entonces que durante el proceso de configuración la eficiencia no se degrada, situación que es beneficiosa para el rendimiento del SMR.

Para configurar un servicio, el *agente de configuración* realiza sucesivas descomposiciones sobre los componentes de dicho servicio, cuando este agente conoce como va a estar configurado un servicio o alguno de sus componentes, no es necesario realizar todo el proceso de configuración. El grado de conocimiento que posee el *agente de configuración* puede ser de mucha ayuda para aumentar la eficiencia de la configuración (ec) de los recursos/servicios, ese grado de conocimiento está relacionado con el criterio de *inteligencia*, el cual consideraremos en el presente estudio. Para ello hemos definido un parámetro ic, a través del cual se expresa el grado de conocimiento que el agente de configuración posee con respecto a las configuraciones que son requeridas durante el proceso de configuración del SMR. El parámetro ic determina el porcentaje de servicios requeridos cuya configuración es conocida por el *agente de configuración*, en los experimentos realizados ic toma el siguiente dominio de valores:

$ic : 5, 10, 25, 50, 80, 100$

Por ejemplo, si ic vale 5, quiere decir que el *agente de configuración* posee información para conocer la configuración del 5 % de los servicios o recursos requeridos.

Realizamos seis experimentos con los distintos valores definidos en el rango de ic. Para dichos experimentos fijamos el número de requerimientos r'' en 8000 como caso estándar, ya que nos interesa observar el comportamiento de ec'' y cc'' sobre una misma instancia de r mientras varía el parámetro ic. Los resultados obtenidos son mostrados en la tabla 5.5.

El número de consultas cc' se obtiene al sumar el número de consultas requeridas para hacer las configuraciones conocidas por el *agente de configuración* (lo llamamos c_1) con el el número de consultas requeridas para hacer las configuraciones no conocidas por el *agente de configuración* (lo llamamos c_2)

$c_1 = r'' = 8000$

$c_2 = (1 - ic) * cc_{8000}$

$cc'' = c_1 + c_2$

donde ca_{8000} es el valor de ca en la tabla 5.1 cuando r vale 8000, en otras palabras, ca_{8000} representa la cantidad de consultas realizadas cuando se necesita más de un intento para realizar la asignación de 8000 requerimientos. Los valores de ea se obtienen de la fórmula 5.2. El comportamiento descrito por los valores presentados en dicha tabla se expresa en las figuras 5.9 y 5.10

	ic						
	0.0	**0.05**	**0.10**	**0.25**	**0.50**	**0.80**	**1.00**
r''	8000	8000	8000	8000	8000	8000	8000
cc''	16745	16308	15871	14559	12373	9749	8000
ec''	0.48	0.49	0.50	0.55	0.65	0.82	1.00

Cuadro 5.5: Experimentos para distintos valores de *ic*

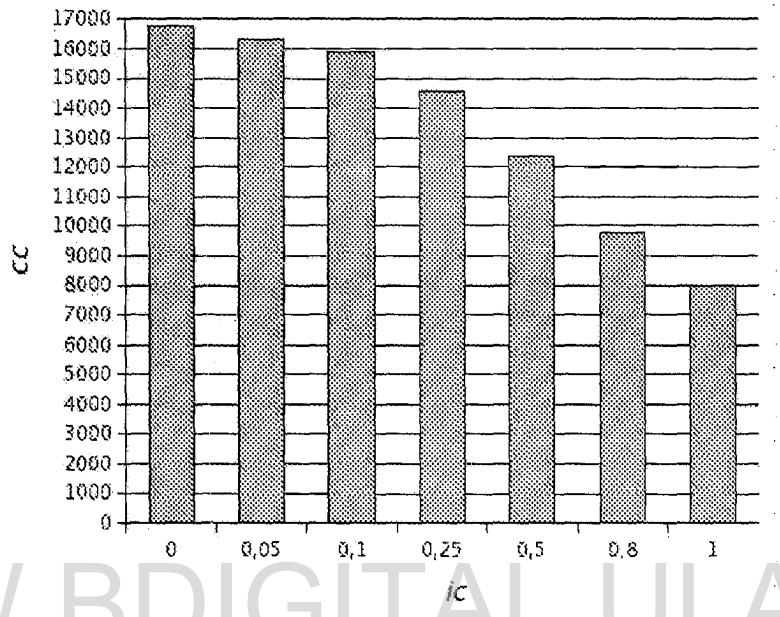

Figura 5.9: Número de consultas en relación a *ic*

Al aplicar técnicas de inteligencia que permitan al *agente de configuración* hacer uso de información almacenada para realizar configuraciones de recursos/servicios se puede reducir el número de consultas, esta situación se refleja en la figura 5.9 donde se puede observa como se logra disminuir el número de consultas (ca) al aumentar el valor del parámetro *ic*. Como es de esperar, cuando el grado de conocimiento es mayor, no se requiere construir las versiones que ya son conocidas, esto ayuda a reducir el número de comunicaciones entre el *agente administrador* y el *agente de configuración*. Así, tenemos que al reducir el número de consultas (ca) se obtiene un aumento en la eficiencia de la configuración (ec), es decir que a medida que aumenta el valor del parámetro *ic* se consigue aumentar la eficiencia de la configuración *ec*, tal como se muestra en la figura 5.10. Vemos entonces que el uso de métodos de inteligencia que permitan explotar la información o el conocimiento manejado por el *agente de configuración*

puede ayudar a mejorar el rendimiento del proceso de configuración de servicios/recursos.

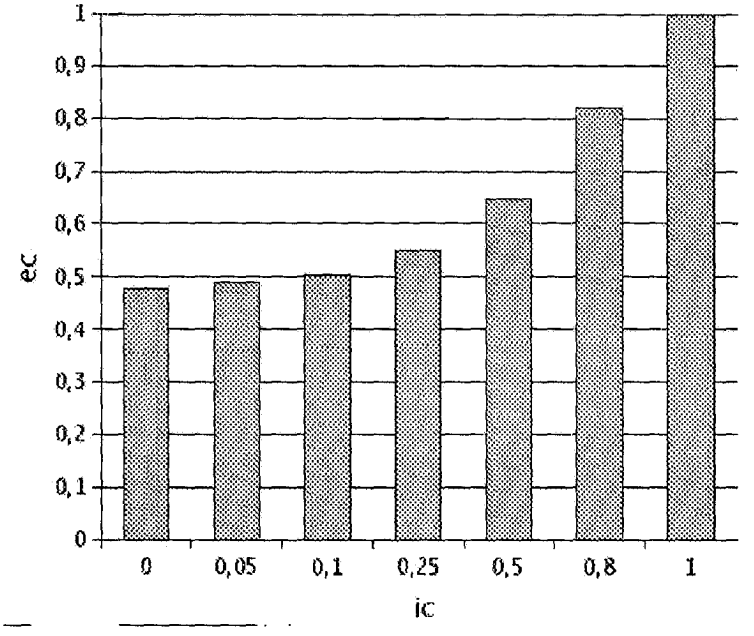

Figura 5.10: Velocidad de configuración en relación a *ic*

5.3.3. Análisis de desempeño para el proceso de búsqueda

En el proceso de búsqueda, un factor que influye determinantemente en el desempeño del SMR es el número de comunicaciones que realiza el *agente localizador* con los Sistemas Manejadores de Repositorios y de Comunidades. Las actividades realizadas por estos sistemas del SOW no han sido simuladas plenamente en este trabajo, sin embargo, en las simulaciones se maneja una métrica que define el número de consultas empleadas durante el proceso de búsqueda, la cual hemos denominado *cb*.

El criterio de inteligencia también tiene importancia para el análisis de rendimiento en el proceso de búsqueda del SMR, pero como hemos mencionado antes, la búsqueda involucra actividades propias de otros sistemas del SOW que no han sido simulados en el presente trabajo, por lo tanto, no hemos incluido un análisis basado en el criterio de inteligencia para el proceso de búsqueda del SMR.

Durante el proceso de búsqueda, el *agente localizador* realiza consultas a los sistemas

manejadores de repositorios y de comunidades. Como es de suponer, se espera que el número de consultas realizadas durante la búsqueda sea bajo para obtener una alta eficiencia de la búsqueda. La eficiencia de la búsqueda promedio viene dada por

$$eb = \frac{r}{cb} \tag{5.7}$$

donde r es el número total de recursos/servicios a ser localizados y cb es el número total de consultas realizadas durante el proceso de búsqueda. Hemos realizado experimentos de donde se han obtenido los valores de cb y eb a partir del siguiente rango de valores para r

$r : 2000, 4000, 6000, 8000, 10000, 12000$

Los resultados obtenidos son mostrados en la tabla 5.6

r	2000	4000	6000	8000	10000	12000
cb	2525	4875	6983	9918	12128	14152
eb	0.79	0.82	0.86	0.81	0.82	0.85

Cuadro 5.6: Experimentos para el proceso de búsqueda

En la tabla 5.6 se observa que el número de consultas (cb) crece regularmente a medida que se incrementa el valor del número de requerimientos (r). Vemos que r y cb tienden a crecer casi en la misma proporción, en consecuencia la eficiencia de la búsqueda eb tiende a mantenerse uniforme. Esto comportamiento se debe a que en este trabajo no ha sido simulada la actividad de los agentes externos a SMR que también intervienen en el proceso de búsqueda, sólo se ha simulado la recepción de las solicitudes de búsqueda que han sido emitidas desde el SMR hacia otros sistemas del SOW, como lo son el Sistema Manejador de Repositorios y el Sistema Manejador de Comunidades. Así, tenemos que los valores de cb acá presentados no corresponden a la simulación del proceso de búsqueda del SOW descrito en [1], sin embargo, nos sirve de referencia para establecer criterios del rendimiento global del SMR.

5.3.4. Desempeño global de SMR

Para examinar el desempeño global del sistema se han definido las dos métricas que se enuncian a continuación.

1. *Sobrecarga*: Es el número total de consultas que realizan los agentes del SMR para satisfacer cierto número de peticiones del usuario. La sobrecarga viene dada por

$$sobrecarga = ca + cb + cc$$

donde ca es el número de consultas realizadas durante el proceso de asignación, cb es el número de consultas realizadas durante el proceso de búsqueda y cc es el número de consultas realizadas durante el proceso de configuración.

2. *Eficacia*: Es la relación que existe entre el número de peticiones atendidas y la sobrecarga que estas generan. La eficacia viene dada por

$$e = \frac{r}{sobrecarga} \tag{5.8}$$

donde r es el número de requerimientos atendidos en cierto período.

A través de los experimentos realizados se han calculado los valores de *eficacia* que se presentan en la tabla 5.7, la cual muestra un compendio de las métricas calculadas durante los procesos de asignación, búsqueda y configuración del SMR.

	Número de peticiones					
Métrica	**2000**	**4000**	**6000**	**8000**	**10000**	**12000**
ca	2163	4791	7707	11339	14292	17441
cc	2103	4315	6282	8745	11087	13526
cb	2525	4875	6983	9918	12128	14152
sobrecarga	6791	13981	20972	30002	37507	45119
e	0,29	0,29	0,29	0,27	0,27	0,27

Cuadro 5.7: Desempeño global de SMR

En la figura 5.11 se observa que la eficacia e se reduce lentamente en relación al número de requerimientos (r), por lo que se presume que la eficacia del SMR no se ve bruscamente afectada por el número de peticiones a procesar. Sin embargo, vemos que la actividad que más afecta la eficacia del sistema es la asignación, ya que a través del proceso de asignación se genera un mayor número de consultas que en los demás procesos, tal como se puede apreciar

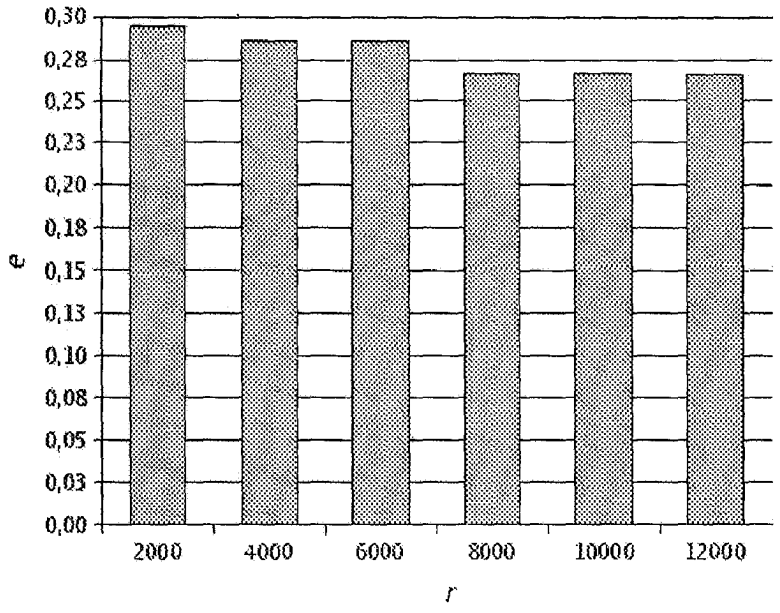

Figura 5.11: Eficacia del SMR

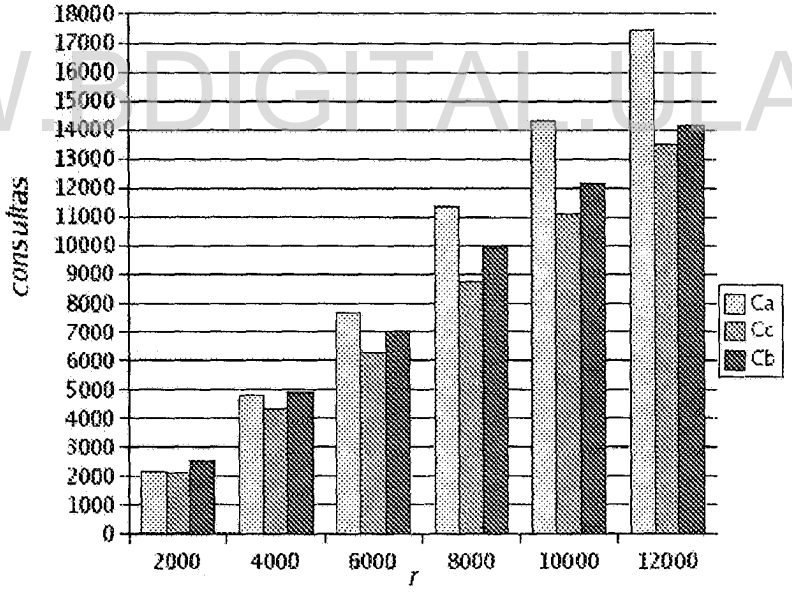

Figura 5.12: Número de consultas en el SMR

en la figura 5.12. Una razón por la cual la asignación contribuye mayormente al aumento de la sobrecarga del sistema, es que para realizar asignaciones en el SMR, el *agente administrador*

actúa dependiendo de otros factores, como la disponibilidad de recursos computacionales. Por otro lado, debemos tener presente que en la actividad de búsqueda intervienen agentes externos al SMR, por lo que esta actividad también pudiera contribuir a la degradación de la eficacia del sistema.

Capítulo 6

Conclusiones y Recomendaciones

El principal propósito de este trabajo ha sido la definición de un Sistema Manejador de Recursos para un SOW. En el diseño de este sistema se han afrontado dos importantes retos, como lo son la escalabilidad y la adaptabilidad del sistema a las situaciones cambiantes que impone el manejo de recursos en la web, esto motivó el uso de métodos dinámicos y mecanismos de inteligencia para poder construir un sistema robusto donde todos sus componentes asumen sus propios roles y cooperan entre si para la solución de problemas.

Como consecuencia de lo anterior, el uso de agentes ha resultado ser una propuesta viable para el desarrollo del SMR, y en función de esta propuesta hemos planteado un diseño del SMR basado en Sistemas Multiagentes. Un diseño íntegro del SOW basado en Sistemas Multiagentes demanda cierta complejidad, ya que se requiere concebir un gran sistema constituido por piezas que implementan mecanismos sofisticados para resolver sus problemas, como una solución a esta circunstancia hemos adoptado la metodología para el desarrollo de sistemas multiagentes MAS-*Common*KADS extendido, obteniendo como resultado una composición organizada de las piezas fundamentales del sistema, y una estandarización en la definición de las operaciones y comunicaciones realizadas en los diferentes módulos del sistema.

Los problemas claves que hemos abordado para el desarrollo del SMR son la volatilidad y la heterogeneidad de los recursos en la web, la solución propuesta para estos problemas consiste en realizar una búsqueda y configuración dínamica de los recursos requeridos. Así, hemos determinado que en el SMR propuesto las principales actividades son la búsqueda, la configuración y la asignación de recursos. Hemos obtenido un diseño que utiliza agentes

autónomos, que pueden estar en capacidad de incorporar nuevas técnicas de búsqueda, configuración y asignación de recursos. Implementamos un prototipo donde se emplearon algunas de estas técnicas, y al evaluar su uso dentro del SMR, se determinó que las mismas influyen determinantemente en el rendimiento del sistema.

La asignación de recursos afecta la eficiencia del sistema, factores como el tiempo de servicio y el ancho de banda pueden contribuir a la reducción de la eficiencia en la actividad de configuración, sin embargo hemos visto que con la incorporación de mecanismos de inteligencia se puede mejorar el desempeño del sistema durante el proceso de asignación. La eficiencia que se pueda lograr depende además de la estrategia de asignación a usar, el diseño propuesto permite la incorporación de nuevas estrategias de asignación.

La configuración de recursos parece ser la actividad que menos afecta el rendimiento del SMR, además de ser una actividad que se beneficia muy bien con la aplicación de mecanismos de inteligencia. El *agente de configuración* es el que tiene más autonomía para realizar sus actividades, ya que durante el proceso de configuración de recursos no es fundamental la intervención de otros agentes o factores externos, como pueden ser, por ejemplo, el tiempo de ejecución, el ancho de banda u otros subsistemas del SOW. Hemos presentado como referencia una estrategia de configuración, pero el SMR está en capacidad de incorporar otras estrategias, por lo que sería conveniente analizar otros algoritmos de configuración que se adapten a las condiciones del SMR propuesto.

En el el diseño del SMR propuesto la responsabilidad de la actividad de búsqueda ha sido repartida entre varios subsistemas del SOW, a través de los experimentos realizados hemos observado que la búsqueda de recursos puede afectar sensiblemente el desempeño SMR en particular, y del SOW de forma global. En un trabajo futuro se evaluarán las particularidades del proceso de búsqueda de servicio como una actividad coordinada entre el Sistema Manejador de Repositorios, el Sistema Manejador de Comunidades y el Sistema Manejador de Recursos.

A través del diseño de SMR planteado, se afrontan problemas que surgen cuando se pretende realizar el manejo de recursos en el ámbito de la web. En el presente trabajo han sido tratados problemas tales como la volatilidad y la heterogeneidad de los recursos en la web,

y las consecuencias han sido alentadoras. Sin embargo, sigue latente el desafío de poder administrar recursos en una estructura tan compleja como la web, estableciendo un orden que permita por ejemplo, realizar búsquedas o asignaciones de recursos con la mayor precisión posible.

Una contribución importante de este trabajo ha sido la concepción de un diseño basado en una arquitectura innovadora que combina aspectos de sistemas de meta-computación con la aplicación de mecanismos de inteligencia dentro de un contexto de sistemas multiagentes. El diseño presentado en este trabajo debe servir como base para trabajos futuros donde se diseñen sofisticados mecanismos de manejo de recursos que permitan robustecer el SMR y donde se definan estrategias que permitan integrar el SMR al SOW.

Bibliografía

[1] J. Aguilar, E. Ferrer, N.Perozo, J. Vizcarrondo. "Arquitectura del Sistema Operativo Web". *Informe Técnico CEMISID*, 2002.

[2] J. Aguilar. *Informe Técnico Agenda Ptróleo*, 2002.

[3] A. Alexandrov, M. Ibel, K. Schauser, C. Scheiman. "SuperWeb: Research Issues in Java-Based Global Computing". *Concurrency: Practice and Experience*, 9(6):535-553, 1997.

[4] G. Babin, P. Kropf, and H. Unger. "A two-level communication protocol for a web operating system (WOS^{TM})". *Proceedings of IEEE Euromicro Workshop on Network Computing*, pp. 934-944, Sweden, 1998.

[5] J. Baldeschwieler, R.Blumofe, and E. Brewe. "ATLAS: An Infrastructure for Global Computing". *Proceedings of 7th ACM Special Interest Group on Operating Systems, European Workshop on System Support for Worldwide Applications*, pp. 165-172, 1996.

[6] A. Baratloo ,M. Karaul, Z. Kedem, and P. Wykoff. "Charlotte: Metacomputing on the Web". *Journal on Future Generation Computer Systems*, 15(5-6):559-570, 1999.

[7] J. Basney and M. Livny. "Managing Network Resources in Condor". *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pp. 298-299, 2000.

`http://www.w3.org/DesignIssues/Axioms.html`. December 1996.

[8] H. Björn. Intelligent Software Agents On The Internet: An Inventory Of Currently Offered Funcionality In The Information Society And A Prediction Of (Near) Future Developments'. *Tilburg University*, The Netherlands, http://www.hermans.org/agents.

[9] A. H. Bond and L. Gasser, editores. Readings in Distributed Artificial Intelligence" *Morgan Kaufmann*, 1988.

[10] N.Camiel, S. London, N. Nisan, and O.Regev. "Globally Distributed Computations Over the Internet - the Popcorn Projec". *Proceedings of the International Conference on Distributed Computing Systems*, pp. 592-601, 1998.

[11] H. Casanova and J. Dongarra. "NetSolve: A Netwok-Enabled Server for Solving Computational Science Problems". *International Journal of Supercomputer Applications and High Performance Computing*, 3(11):212-223, 1997.

[12] B. Chaib-draa, B. Moulin and R. Mandiau, and P. Millot. Trends in Distributed Artificial Intelligence" *Artificial Intelligence Review*, 6:35-66, 1992.

[13] B.O. Christiansen, P Cappello, M. F. Ionescu, M. O. Neary, K E. Schauuser, and D. Wu. "Javelin: Internet-Based Parallel Computing Using JAVA". *Concurrency: Practice and Experience*, 9(11):1139-1160, 1997.

[14] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith and S. Tuecke. "A Resource Management Architecture for Metacomputing Systems". *Lecture Notes in Computer Science*, 1459:62-82, 1998.

[15] Condor Team. "Condor Manual". *University of Wisconsin*, Madison, 1999.

[16] I. Durdanović and H. Kleine Büning. "An algorithm for resource-based configuration". *Technical Report tr-rsfb-96-022, Department of Computer Science, University of Paderborn*, 1996.

[17] S. I. Feldman. "Make - A Program for Maintaining Computer Programs". *Software: Practice and Experience*, 9(4):255-265, 1979.

[18] I. Foster and C. Kesselman. "Globus: A Metacomputing Infrastructure Toolkit". *Supercomputer Applications*, 11(2):115-128, 1998.

[19] Grid Computing Infoware (Info Centre) `http://www.gridcomputing.com/`

[20] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds. "A Synopsis of the Legion Project." *Technical Report CS-94-20, University of Virginia*, 1994.

[21] C. A. Iglesias. "Definiciń de una Metodología para el Desarrollo de Sistemas Multiagentes". *Tesis Doctoral*, Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, 1998.

[22] C. Iglesias Fundamentos de los Agentes Inteligentes". *Informe Tcnico* UPM/DIT/GSI-19/97. Departamento de Ingeniera de Sistemas Telemticos, Universidad Politcnica de Madrid.

[23] J. P. Ignizio, "Introduction to Expert Systems: The Development and Implementation of Rule-Based Expert Systems", McGraw-Hill, 1991.

[24] I. Jacobson, M. Christerson, P. Jonsson, and Övergaard. "Object-Oriented Software Egineering. A Use Case Driven Approach". *ACM Press*, 1992.

[25] Klügl F. and Puppe F. "The Multi-Agent Simulation Environment SeSAm". Technical Report TR-RI-98-194, Reihe Informatik, Universit at Paderborn, 1998.

[26] O. Krone, S. Schubiger. "WebRes: Towards a Web Operating System". *Kommunikation in Verteilten Systemen*, pp. 418-429, 1999.

[27] P. Kropf Sainte-Foy. "Overview of the WOS Poject". *Advanced Simulation Technologies Conferences, High Performance Computing*, pp. 350-356, 1999.

[28] P. Kropf, H. Unger, and G. Babin. "WOS: an Internet Computing Environment". *Proceedings of the Workshop on Ubiquitous Computing, PACT 2000 (IEEE International Conference on Parallel Architectures and Compilation Techniques)*, pp. 14-22, 2000.

[29] S.B. Lamine and J. Plaice. "Problems of Computing on the Web". *Proceedings of the 1997 High Performance Computing Symposium*, pp. 296-301, 1997.

[30] S.B. Lamine and J. Plaice. "Simultaneous Multiple Versions: The Key to the WOS". *Proceedings of Distributed Computing on the Web (DCW 98)*, pp. 122-128, 1998.

[31] B. Langley, M. Paolucci, and Sycara, K. "Discovery of Infrastructure in Multi-Agent Systems". *Actas de conferencia. Agents 2001 Workshop on Infrastructure for Agents, MAS, and Scalable MAS.* pp.82-90, Barcelona - España, 2001.

[32] Microsoft Corporation. "Microsoft MS-DOS Operating System Version 5.0 - User's Guide and Reference", Microsoft Corporation. 1991.

[33] S. Morgan. "Jini to the rescue". *IEEE Spectrum*, 37(4):44-49, 2000.

[34] B. Moulin and B. Chaib-draa. Fundationns of Distributed Artificial Intelligence" *John Wiley & Sons*, 1996.

[35] H. S. Nwana, Hyacinth S. Software Agents: An Overview". *Knowledge Engineering Review.* 11(3):205-244, 1996.

[36] M. Parameswaran, A. Susarla, and A. B. Whinston. "P2P Networking: An Information Sharing Alternative". *IEEE Computer*, 34(7), 2001.

[37] J. L. Peterson and A. Silberschatz. "Operating System Concepts". *Addison-Wesley Publishing Company*, 1986.

[38] J. Plaice and S.B Lamine. "Eduction: a general model for computing". *In Intetional Proramming II. World Scientific*, Singapore, 1997.

[39] J. Plaice and W.W. Wadge. "A New Approach to Version Control". *IEEE Transaction of Software Engineering*, 19(3):268-276, 1993.

[40] C. Potts, K. Takahashi and A. Anton. "Inquiry-Based Scenario Analysis of System Requirements". *Technical Report* GIT-CC-94/14, Georgia Institute of Technology, 1994.

[41] F. Reynolds. "Evolving an Operating System for the Web". *IEEE Computer*, 29(9):90-92, 1996.

[42] J. Rumbaugh. "OMT: The Development Process". *Journal of Object Oriented Programming*, 8(2):8-16, 1995.

[43] S. Smallen, H. Casanova and F. Berman. "Applying Scheduling and Tuning to On-line Parallel Tomography". *to appear in Scientific Programming*, 2002.

[44] W. Stallings. "Sistemas Operativos". Prentice-Hall, 2001.

[45] P. Stone and M. Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective". *IEEE Transactions on Knowledge and Data Engineering*, 1996.

[46] A.Tanenbaum. "Modern Operating Systems", 2nd ed. Prentice-Hall, 1999.

[47] A. Vahdat, t. Anderson, M. Dahlin , E. Belani, D. Culler, P. Eastham, and C. Yoshikawa. "WebOS: Operating System Services for Wide Area Applications". *In Seventh IEEE Symposium on High Performance Distributed Systems*, pp.52-63, USA, 1998.

[48] M. Van Steen, P. Homburg, and A. S. Tanenbaum. "The Architectural Design of Globe: A Wide-Area Distributed System". *Technical Report IR-422*, Vrije Universiteit, Amsterdam, 1997.

[49] T. H. Van Vleck. "Multics Features". `http://www.multicians.org/featutes.html`.