UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

"AGENTES PARA LA ADMINISTRACION Y SEGURIDAD DE EQUIPOS DE COMPUTO EN RED"

T E S I S

QUE PARA OBTENER EL GRADO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N :

NORBERTO JESUS ORTIGOZA MARQUEZ

ANDRES PINEDA RUIZ

DIRECTOR: ING. MARIO RODRIGUEZ MANZANERA

CODIRECTOR: ING. JUAN JOSE CARREON GRANADOS

MEXICO, D. F. MARZO DE 1997

Agradecimientos

De Norberto:

A mis padres Norberto Ortigoza Toledo y Hermelinda Márquez Hernández por su cariño y ayuda incondicional durante toda mi vida.

De Andrés:

A mis padres Wiliulfo Pineda Reyes y Alicia Ruiz Sánchez por brindarme la oportunidad de culminar una etapa más en mi vida y por su amor.

A mis hermanos porque puedo contar con ellos en todo momento.

De Norberto y Andrés:

Al Ing. Mario Rodríguez Manzanera por apoyarnos y asesorarnos durante la elaboración de este trabajo, y por brindarnos su valiosa amistad.

Al Ing. Juan José Carreón Granados por su apoyo incondicional durante nuestra estancia en la Facultad de Ingeniería, y por formar en nosotros el gusto por la investigación y el deseo de superación

A la Dra. Hanna Oktaba por permitir complementar nuestro desarrollo académico en la Maestría en Ciencias de la Computación, y por sus valiosos comentarios a este trabajo.

A todos nuestros amigos.

Contenido

Introducción

El creciente uso de redes de computadoras en las empresas así como el surgimiento de nuevos conceptos como el de Intranets pone ante los usuarios nuevas e innovadoras formas de trabajo, también impone nuevos retos para los administradores de estas redes. Por otra parte Internet ha provocado la evolución y consolidación de nuevas tecnologías que pueden ser empleadas para solucionar algunos de los problemas que implica el tener redes con diferentes equipos ubicados inclusive en diferentes partes del mundo.

Este trabajo tiene por objetivo mostrar el uso de dos de las tecnologías que han mostrado un creciente desarrollo en los últimos años para la creación de un sistema de administración de equipos de cómputo.

En este trabajo no se pretende dar una solución global a todos los problemas a los que se puede enfrentar una o varias personas que tienen a su cargo la labor de administrar uno o varios centros de cómputo. Lo que se intenta es mostrar qué tan factible es el uso de estas tecnologías en la creación de sistemas complejos creando un sistema de administración de equipo de cómputo que abarca algunas de las tareas que absorben gran parte del tiempo del administrador de sistemas.

El capítulo 1 presenta el desarrollo que han tenido los centros de cómputo en la forma que utilizan e interconectan el equipo existente, dando a conocer los diferentes problemas a los que se enfrenta el administrador en cada una de las etapas que se describen: desde los mainframes hasta el surgimiento más reciente "las Intranets". Se describen las limitantes que existen con algunas herramientas de administración y se expone la necesidad de contar con una herramienta multiplataforma y distribuida para resolver problemas actuales.

Dado que en nuestro desarrollo se utiliza Java, en el capítulo 2 se presentan los puntos principales del lenguaje Java, la evolución que ha tenido y el impacto que puede representar en la creación de sistemas distribuidos y en general de sistemas que trabajan en redes de computadoras. Se muestran algunas de sus características que lo hacen atractivo para la implementación de un sistema y el crecimiento que tendrá este lenguaje en el futuro. También se menciona la importancia que tiene la programación orientada a objetos para el modelado de sistemas, particularmente los que hacen uso de la teoría de agentes.

El capítulo 3 esta dedicado a un breve estudio sobre la tecnología de agentes en el cual se menciona la forma en la que ha evolucionado, se da una definición de lo que se considera un agente, se remarcan las características que los hacen diferentes de otro tipo de software, se presenta una clasificación de los mismos, se explica la forma en que éstos funcionan, se hace referencia a las investigaciones que actualmente se están haciendo sobre este tema y finalmente se explica la razón por la cual esta tecnología se muestra adecuada para la solución al problema de la administración de sistemas de cómputo.

Durante el capítulo 4 se hace una descripción de las nuevas extensiones al lenguaje Java para cómputo distribuido que permiten la creación de agentes móviles. También se describen dos de los principales frameworks hechos en Java para la creación de agentes, se plantean cada una de sus ventajas y desventajas, así como las áreas en las que cada uno de los frameworks encaja mejor.

En el capítulo 5 se describe todo el desarrollo del sistema de administración, su análisis, diseño, prototipo e implementación final. Se muestran y explican cada una de las clases y paquetes creados, así como los diagramas de dichas clases. También se muestran a detalle las clases utilizadas del framework y su importancia en el diseño del sistema.

Al final se presentan las conclusiones a las que se llegaron durante y al final del trabajo, se hace referencia a los problemas a los que nos enfrentamos y la forma en que se solucionaron. Se muestran algunas pruebas realizadas al sistema para analizar y cuantificar su rendimiento y se trata sobre los recursos que demanda. Finalmente se presentan en los apéndices el API del Aglets Workbech así como el API de nuestro sistema.

Capítulo 1
Antecedentes

Hace algunos años las grandes empresas contaban con su equipo mainframe y un gran número de "terminales tontas" que servían como medio de acceso para el equipo principal. Todo el proceso se realizaba en el mainframe y las terminales sólo se utilizaban como dispositivos de entrada/salida, es decir, el usuario se encontraba ante una arquitectura centralizada. En ese entonces la idea de una red global conformada por una gran cantidad de redes locales y públicas por la cual se pudieran enviar y recibir mensajes o realizar transacciones a nivel mundial no existía o sólo se encontraba en la imaginación de algunos investigadores. Ver figura 1.

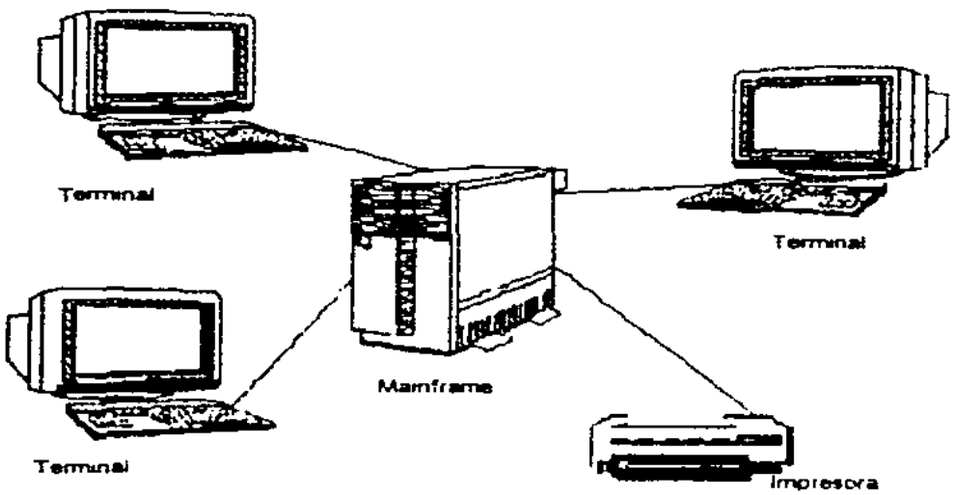


Figura 1 Arquitectura de cómputo centralizada

En la actualidad es posible compartir los recursos de cómputo que se encuentran distribuidos en distintas partes de una oficina, de un edificio, de una ciudad, o inclusive del mundo sin tener que trasladarse para poder hacer uso de ellos. Esto se debe a que las nuevas tecnologías se están desarrollando tan rápido, que en un abrir y cerrar de ojos nos encontramos con nuevos equipos en hardware y con nuevos sistemas en software que poseen características que facilitan tanto las comunicaciones como la forma de compartir recursos a larga distancia, así como nuevas empresas que nos proporcionan servicios de interconectividad con redes públicas de datos y acceso a grandes bancos de información en todo el mundo. Ver figura 2.

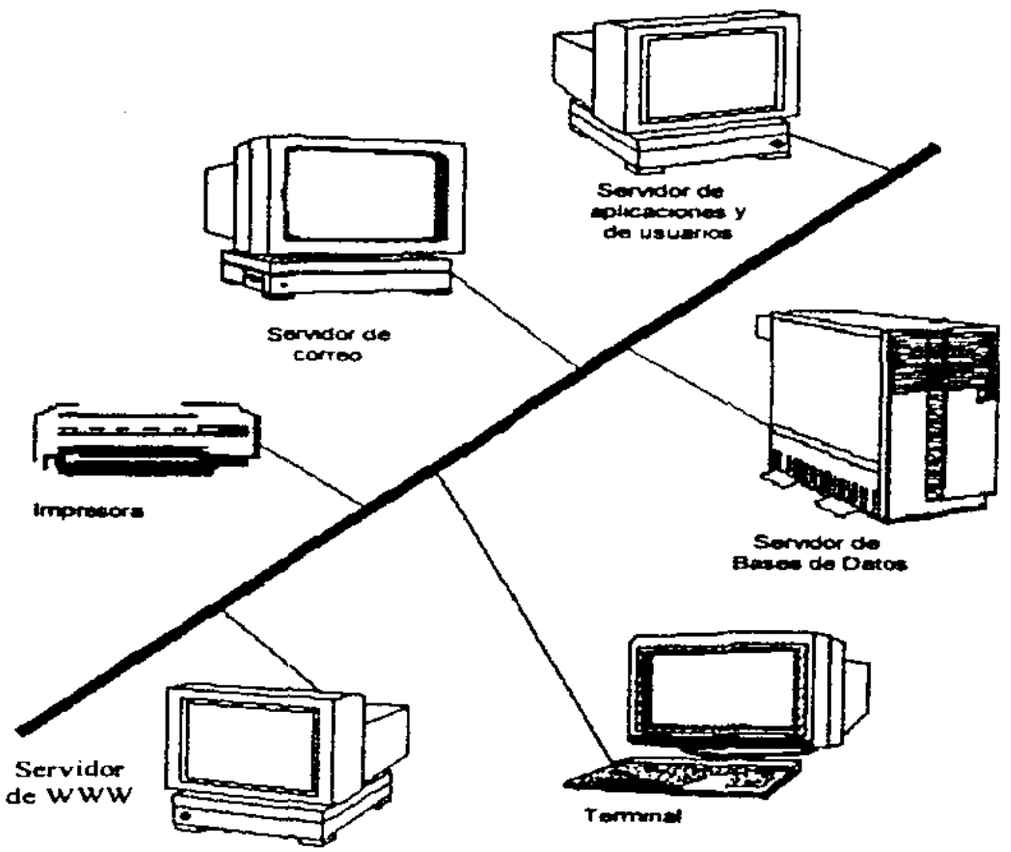


Figura 2 Arquitectura de cómputo descentralizada

Con el creciente uso de redes locales (LANs), Internet y de intranets dentro de las organizaciones, se han presentado ante sus usuarios nuevas e interesantes formas de obtener tanto información como comunicación con otras personas dentro y fuera de la organización a la que pertenecen. Así, se ha creado una nueva manera de proporcionar nuevos y mejores servicios a los clientes de las empresas. Sin embargo, aunque esto es ya una realidad aún se encuentra en su etapa inicial con la esperanza de que con el tiempo se constituya como parte integral de la operación de las organizaciones.

Intranets

Una intranet puede considerarse como el conjunto de equipos que conforman a las WANs y LANs, a las arquitecturas cliente/servidor y a las redes de PCs que se han estado usando en las organizaciones para realizar su trabajo, mejorar su eficiencia, colaboración, y/o comunicación con otros. Ver figura 3.

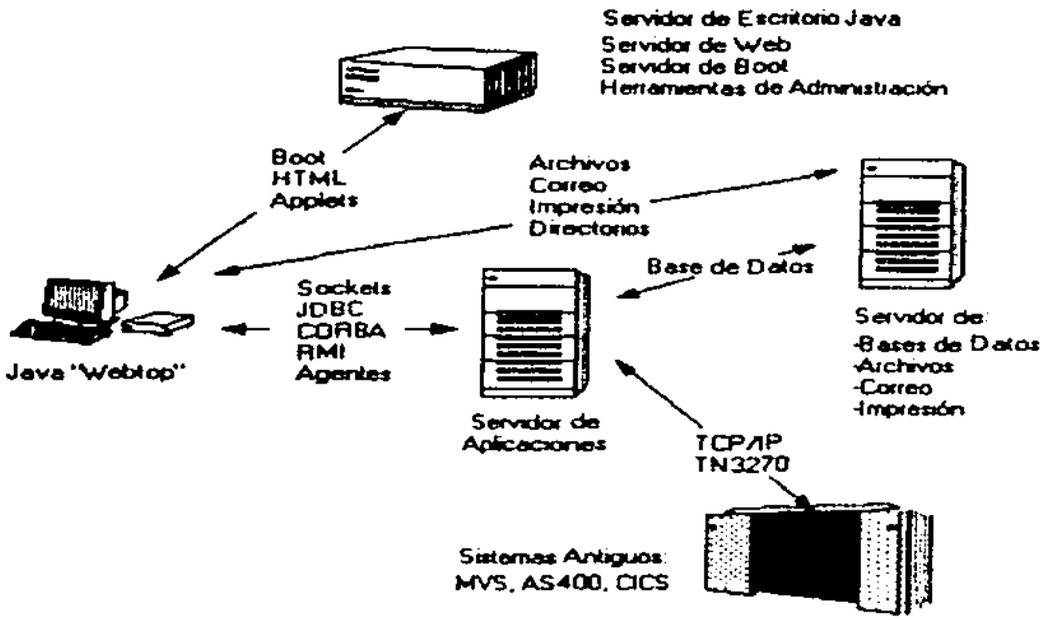


Figura 3 Intranet

No obstante, una intranet no se logra simplemente con poner equipos, software y comunicaciones a trabajar en conjunto. Intranet es un sistema de información interna de las empresas basado en la tecnología de Internet, servicios de WWW, protocolos de comunicación TCP/IP y HTTP, y publicidad en HTML. Es una tecnología que permite a una organización definirse como una entidad completa, un grupo, una familia donde cada quien conoce sus roles y todos trabajan para mejorar la organización.

Lo anterior se logra al identificar y comunicar las misiones de cada área de negocio, así como sus objetivos, sus procesos, sus relaciones, sus interacciones, su infraestructura, sus proyectos, sus agendas, y sus presupuestos, en una misma y simple interfaz que cada uno usa para añadir valor mediante la retroalimentación en línea. Una intranet representa la inteligencia de la organización. El propósito de esa inteligencia es organizar el escritorio de cada individuo con un mínimo de costo, tiempo y esfuerzo para hacer más productiva a la organización, más eficiente en sus costos, y así ser más competitivos en el mercado.

Un problema que se venía presentando a los negocios era el derivado de la utilización de sistemas propietarios, lo cual impedía el compartir información con equipos y sistemas que no fueran del mismo proveedor. Este problema ha sido

casi superado con la aparición de los sistemas abiertos, en donde con base en estándares de producción y construcción, se ha intentado liberar al usuario de la

necesidad de contar con únicamente un proveedor o un solo tipo de equipo, pudiendo entrar en un nuevo mercado de competencia al poder intercalar equipos de diversa procedencia con software de diferentes proveedores, pero en particular al poder interconectarse a través de redes que, utilizando protocolos estándares, se muestran transparentes a sus deseos de transferir y compartir información con usuarios remotos en línea.

Con una intranet, el usuario tiene acceso a toda la información, programas de aplicación, datos y procesos de otros usuarios por medio de una misma ventana, o visualizador. Uno de los objetivos de las intranets es reducir al mínimo la necesidad de convertir la información a diferentes formatos, o bien tener que esperar que los administradores o programadores codifiquen los nuevos sistemas para poder compartirlos, con lo cual se reducirá la posibilidad de perder oportunidades o el tener que cancelar y evitar establecer negocios con compañías debido a que su tecnología es diferente a la propia. En vez de eso, una intranet interconecta al personal de la organización en conjunto, mientras continúan usando el software adquirido con anterioridad.

Esto representa una oportunidad para que las organizaciones se definan y se muestren ante todas las organizaciones conectadas a la red, y específicamente a los usuarios internos. Si todos conocen los objetivos de la compañía, la visión estratégica de la empresa y los principios que la guían, y además quiénes son sus clientes y socios de negocio, entonces podrán enfocarse claramente sobre sus propias contribuciones para la organización.

La capacidad del personal de utilizar correo electrónico, distribuir información relacionada con su trabajo a cada uno de los demás usuarios por medio de su computadora y el permitir que la totalidad del personal y el equipo de desarrollo de software ya no se encuentren en un mismo lugar físicamente, son sólo algunas nuevas facilidades que se obtienen con el uso de intranets. Por otra parte, el explosivo desarrollo que ha tenido el WWW (World Wide Web) como herramienta de acceso a diversos servicios de información dentro de muchas organizaciones ha provocado, entre otras cosas, que organizaciones alejadas del área de las comunicaciones se interesen no solamente en su uso, sino en el establecimiento de unidades propias (servidores de WWW), contando con grandes equipos servidores y promoviendo su utilización, tanto interna con sus empleados, como externa con sus clientes, produciéndose con ello un incremento tanto de equipos servidores interconectados a Internet como de usuarios y de requerimientos de interconexión y de medios de comunicación. La razón de ello es que, el WWW se presenta como una alternativa en la forma de hacer negocios de forma electrónica, dinámica, interesante, fácil de usar y de mantener. Ver figura 4.

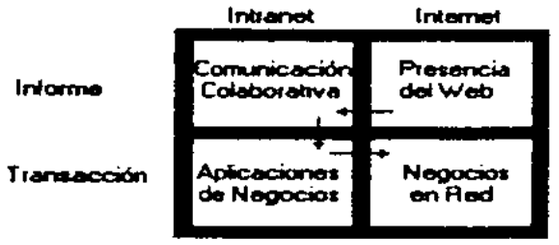


Figura 4 Marco para redes de negocios

Problemas que se presentan en la administración de sistemas de información actualmente

Tan rápido como la tecnología se desarrolla, la necesidad de los usuarios de hacer uso de ésta se acrecienta, pues la demanda y su interés en la misma es cada vez mayor, ya sea por estar a la vanguardia o por encontrar una mejor alternativa a la solución de sus problemas. Los usuarios de la tecnología requieren de mejores servicios para el manejo de su propia información y a la vez de mejores y más potentes recursos de software y de hardware. Sin embargo, no sólo se necesita de la adquisición de equipos o de herramientas que permitan aprovecharlos, también es necesario llevar un riguroso control de éstos, conociendo en todo momento con qué se cuenta para poder brindar un mejor servicio a los usuarios, ahorrar dinero y esfuerzo.

Como resultado de lo anterior, al administrador de las redes y sistemas se le presentan problemas cada vez más difíciles de resolver. El incremento de conexiones, equipos, usuarios, etc. crea nuevos problemas dentro de las grandes corporaciones, debido a que se trata de sistemas descentralizados, y en algunos casos con instalaciones en diferentes ciudades, trabajando bajo un ambiente cliente/servidor. Las dudas en cuanto a uso, servicios, accesos, etc. por parte de los usuarios aumentan considerablemente puesto que no están acostumbrados a trabajar en red, así, es necesario proporcionarles cierta capacitación para que puedan trabajar de una forma eficiente utilizando esta nueva tecnología en lugar de proporcionarles más problemas. Actualmente es necesario que el administrador tenga que ir físicamente al lugar del problema para saber qué está pasando con la computadora del usuario. Esto repercute grandemente en pérdida de dinero para las organizaciones y pérdida de tiempo para el administrador; ya que por lo regular los problemas son menores y fáciles de resolver.

Estos problemas se acrecientan cuando se tiene que administrar no sólo un equipo central como se hacía hace algunos años en donde la información se concentraba en un solo lugar, sino que ahora al tener una cantidad enorme de PCs y workstations, muchas veces coexistiendo en un ambiente heterogéneo, la información al igual que el control se encuentra distribuida sobre la red

incluyéndose en esto, por supuesto, a los antiguos equipos mainframe, los cuales tienen que ser configurados y administrados eficientemente.

Otro problema, derivado del anterior, es la necesidad de contar con un grupo más grande de personas especializadas que puedan resolver cualquier problema que pueda presentarse con las instalaciones o con el funcionamiento de los equipos, pues actualmente, según estadísticas del USENIX SAGE (una de las organizaciones más importantes a nivel mundial en el área de workstations y UNIX), el promedio de estaciones de trabajo atendidas por un administrador es de casi 50, cuando debería ser aproximadamente de 20 estaciones, lo cual impacta en forma considerable a la empresa en sus costos operativos. La gente que posee tanto las características como el perfil de un administrador, por lo general no tiene la disponibilidad para trasladarse de un lugar a otro para llevar a cabo las labores de administración y hay instituciones o empresas que no tienen el presupuesto para tener varios administradores o uno al menos para cada una de sus unidades de sistematización de datos, ni para capacitar en forma adecuada a los que ya tienen a fin de que estén actualizados de los cambios que se presentan en el ámbito de redes y sistemas.

También es común encontrar dentro de las empresas, principalmente en las grandes, la falta de control del equipo y del software del que se dispone (inventario). En ocasiones el problema se presenta debido a que no existe congruencia entre lo que se supone debe estar instalado y lo que realmente está, ya que cuando el administrador compara su registro manual, o sea los resguardos, contra lo instalado descubre que existen diferencias notables: el usuario, o bien ha instalado otro tipo de software, o ha cambiado (sino es que eliminado) archivos de configuración, o la cantidad de memoria RAM no es la registrada, etc. Este problema se da principalmente por que el administrador tiene que realizar el control de inventarios de forma manual lo que implica una gran inversión en tiempo y gente.

Así, para la solución de estos nuevos problemas, es de gran importancia que el administrador pueda contar con las herramientas adecuadas para realizar la administración de la red y del equipo conectado a ella, buscando siempre el menor costo y la forma eficiente, confiable y segura de hacerlo. Además de con ello, tratar de evitar en lo posible fallas o caídas de los sistemas así como asegurar su funcionamiento y recuperación oportuna.

El sistema de administración que apoya al administrador debe ser lo suficientemente robusto como para seguir funcionando a pesar de que alguna parte de la red quede incomunicada, lo cual presupone que el sistema de monitoreo de fallas tenga una amplia cobertura, y esté fuertemente conectado y distribuido a lo largo y ancho de la organización.

Para cada uno de los problemas comentados las empresas han intentado utilizar un conjunto de soluciones basadas en diversas herramientas y tecnologías. Sin embargo, existe un amplio conjunto de limitaciones al intentar su aplicación, derivado de que las soluciones han sido desarrolladas y diseñadas en forma

independiente, es decir, una solución para cada problema, careciendo de una solución integral.

Algunas de las limitaciones que presentan dichas herramientas se enuncian a continuación:

- Las herramientas sólo son funcionales para algún tipo de plataforma (entendiéndose por ésta, la arquitectura del sistema, y el sistema operativo que utiliza). Desafortunadamente la gente o empresa que las desarrolla, muchas veces no contempla su implementación en más de una o todas las plataformas disponibles para su ejecución, con lo cual se limita su uso a sólo algunos sistemas.

- Poca o nula integración con otras herramientas. Muchas de las herramientas que brindan soluciones diferentes en apoyo a la administración de sistemas y de redes carecen de un mecanismo que les permita aprovechar las capacidades de otras herramientas o brindar un medio a través del cual otras herramientas puedan hacer uso de su información para procesarla y obtener resultados.

- Las herramientas y la información se concentran en un servidor principal, el cual al ser deshabilitado deja la totalidad del sistema sin monitoreo. Muchas veces el administrador, por razones propias o ajenas, se ve en la necesidad de dar de baja alguno de los sistemas que tiene a su cargo, por lo que si las herramientas que posee dependen de una máquina que es dada de baja, el administrador puede perder el control de lo que ocurre con esa y otras máquinas.

- Al interactuar las herramientas con la totalidad de la red producen una sobrecarga en los elementos de interconexión con el servidor. Dado que las tareas de monitoreo de recursos del sistema, y de la interconectividad de la red demandan demasiados recursos en el sistema donde se encuentra instalada la herramienta, el desempeño de los sistemas se puede ver afectado tanto en procesamiento como en velocidad de acceso a la red.

Actualmente las herramientas de administración han sido clasificadas de la siguientes forma: herramientas para el administrador de sistemas y herramientas para el administrador de la red. Donde el administrador de sistemas es responsable de la instalación y configuración de los equipos, del control de licencias de software, de la verificación del rendimiento del hardware y responsable de sus resguardos, mientras que el administrador de redes es responsable del monitoreo del tráfico de la red, de la instalación y configuración de equipos de comunicaciones y del software de la red, así como del control de inventarios, garantías y contratos del equipo. Esta forma de clasificación tiene algunos inconvenientes; ya que en muchas ocasiones una o varias personas realizan conjuntamente las tareas de administración de sistemas y redes, quedando esta división completamente difusa.

Otro aspecto importante que un administrador de redes y de sistemas debe tener en cuenta es el de la seguridad de los sistemas, pues debido a la gran cantidad de equipos conectados a una red actualmente, los riesgos que corren, han aumentado considerablemente. Existen diversos tipos de seguridad que un administrador debe conocer. A continuación se mencionan algunos de los tipos relacionándolos a:

- Privacidad. Implica limitar el acceso a la información a personas que no tienen permiso explícitamente brindado por el dueño de esa información. Incluye no sólo proteger la información que se considere privada sino también aquélla que pareciera inofensiva por sí misma pero que puede ser usada para accesar a la información confidencial.

- Integridad de los datos. Es importante evitar que la información (incluyendo programas) sean borrados o alterados de alguna manera sin el permiso del propietario de la información. La información que debe ser protegida también incluye a aquéllas formas que pudieran no parecer tan obvias como registros de cuentas, cintas de respaldo, fechas de creación de archivos, y documentación.

- Disponibilidad. El administrador es responsable de proteger los servicios para que no sean degradados o hacerlos no disponibles sin autorización. Si el sistema no está disponible cuando un usuario autorizado lo necesita, eso puede ser tan malo como si la información hubiera sido borrada.

- Consistencia. El sistema debe comportarse como es esperado por los usuarios autorizados. Si el software o el hardware de repente comienza a comportarse completamente diferente de la forma en que generalmente lo hace, especialmente después de alguna actualización o de haberse corregido algún error, esto podría ser un desastre.

- Aislamiento. Es necesario regular el acceso al sistema. Si se encuentra a algún individuo o software desconocido y sin autorización, esto puede crear un problema. Si esto sucede se debe averiguar la forma a través de la que tuvo acceso y lo que podría haber hecho, y quién o qué más ha accesado al sistema. Recuperarse de tales situaciones podría implicar una gran cantidad de tiempo y esfuerzo reconstruir y reinstalar el sistema, y después verificar que nada importante ha sido cambiado.

- Auditoría. De la misma manera en que son preocupantes las acciones llevadas a cabo por los usuarios no autorizados, los usuarios autorizados algunas veces cometen errores, o cometen actos maliciosos. En casos como estos, se debe determinar lo que fue hecho, por quién, y qué fue afectado. La única manera de llevar a cabo esto es a través de un registro de actividades (que no pueda ser alterado) en el sistema que identifique positivamente los actores y las acciones implicadas. En algunas aplicaciones críticas, el rastro de auditoría podría ser lo suficientemente completo como para permitir deshacer las operaciones y ayudar a restaurar el sistema a un estado correcto.

Es notable que la labor del administrador resulta demasiado compleja si no cuenta con las herramientas adecuadas para poder mantener trabajando los sistemas y funcionando correctamente la red a su cargo, o si las herramientas con las que cuenta no tienen la capacidad de cubrir varios de los aspectos ya mencionados. Más aún cuando tiene a su cargo una gran cantidad de equipo bajo su responsabilidad y dicho equipo no se encuentra en un mismo sitio.

Por lo anterior es necesario hacer una revisión de las nuevas tecnologías que creemos nos pueden acercar a una solución a uno o varios de los problemas mencionados a lo largo de este capítulo. En el siguiente capítulo se tratará sobre una de las tecnologías que actualmente está generando una nueva manera de proporcionar soluciones a problemas que implican el manejo de sistemas distribuidos: Java.

Capítulo 2

Java y Programación Orientada a Objetos.

Internet ha sido la impulsora del concepto de intranets y de toda una revolución en la forma de distribuir información y realizar las comunicaciones dentro de una red local. Esto a su vez ha provocado el desarrollo de nuevas tecnologías tales como la de "agentes" y el lenguaje Java. En las cuales se basa la integración del hardware y del software en Internet. Este capitulo está dedicado a explicar que es Java y su importancia en la administración de redes.

Evolución de Java

Inicialmente Java comenzó como un lenguaje de programación que facilitaría el desarrollo de aplicaciones e interfaces para dispositivos electrodomésticos tales como televisores, hornos de microondas, etc. Sin embargo, con el crecimiento que tuvo el uso de Internet gracias al surgimiento del WWW y los visualizadores gráficos, se hizo un replanteamiento del uso de este lenguaje para que fuera utilizado como el lenguaje de programación ideal para Internet. Desde su concepción Java fue diseñado para trabajar en red, por lo que fueron diseñadas un conjunto de rutinas especializadas para conformar bibliotecas que permiten la rápida creación de aplicaciones cliente/servidor, utilizando como protocolo de transporte TCP. Java tuvo una rápida aceptación entre la comunidad de Internet; ya que parecía ser la respuesta a muchos de los problemas que se venían arrastrando tales como portabilidad y falta de estándares. Además de que dicho lenguaje fue distribuido de forma gratuita a los usuarios ordinarios. Tal fue su aceptación que corporaciones tan grandes e importantes como Microsoft, Netscape, Silicon Graphics, IBM, HP, entre otras, empezaron a adquirir licencias para usar esta tecnología en sus propias plataformas y aplicaciones.

Netscape fue la primera empresa que incluyo Java en su visualizador, permitiendo la ejecución de pequeñas aplicaciones llamadas applets. Estos applets son programas creados con Java que proporcionan una mayor interacción con la persona que visualiza una página en el visualizador. Siendo ésta la principal causa que impulso a la comunidad al uso de Java en Internet.

Este crecimiento acelerado provocó que el término Java ya no se asocie únicamente a un lenguaje de programación, sino a todo un conjunto de sistemas de hardware, y software (sistema operativo y lenguaje) que están pensados para explotar al máximo el concepto de intranets.

En el caso del sistema operativo tenemos que la compañía SUN ha diseñado un nuevo sistema operativo pequeño y eficiente que puede o no incluir una máquina virtual para la ejecución de programas hechos en Java (esto depende de si el sistema operativo es ejecutado en una máquina Java o en alguna

arquitectura diferente), y un visualizador de WWW desde donde se pueden accesar todas las aplicaciones que necesita el usuario final (procesador de palabras, hoja de cálculo, creador de presentaciones, administradores de bases de datos, etc.).

En el lado de hardware se ha desarrollado hardware específico para que las aplicaciones hechas en Java se ejecuten eficientemente. Actualmente existen tres versiones de lo que se conoce como la máquina Java: microjava, picojava y ultrajava. Cada una de estas implementaciones incluye una interfaz de red, una memoria ROM de 4 megas así como 4 megas de RAM. En la memoria ROM se encuentra instalado el sistema operativo JavaOS, esto hace que el sistema operativo sea cargado rápidamente en memoria y la máquina inmediatamente entre en funcionamiento.

El lenguaje Java por sí mismo, ha madurado cada vez más incrementando considerablemente el conjunto de bibliotecas con el que se dispone actualmente. Se han desarrollado bibliotecas de clases para controlar dispositivos electrónicos, para accesar a bases de datos, para telecomunicaciones, para VRML, para multimedios, etc. También ha crecido el numero de plataformas en las cuales pueden ser ejecutados los programas en Java.

Este lenguaje se presenta como una buena opción para la implementación de otras tecnologías como la de agentes, y por lo tanto de sistemas de administración de datos, y de búsqueda y filtrado de información, debido a que proporciona grandes facilidades para desarrollar sistemas distribuidos, además, el lenguaje es sumamente portable. Por portable entendemos que el sistema que desarrollemos en Java podrá trabajar sin ningún cambio en equipos UNIX (Solaris, NextStep, HP-UX, Irix), PC's (Windows 95, Windows NT), equipos Macintosh y Mainframes. Esto contrasta notablemente con otros lenguajes de programación como C, C++, Pascal y VisualBasic; ya que con estos lenguajes resulta muy difícil portar un programa de una plataforma a otra, principalmente por incompatibilidad con las bibliotecas, con el administrador de ventanas (interfaz gráfica) o debido a que el compilador del lenguaje como tal no está disponible en todas las plataformas.

Esta portabilidad es de gran utilidad ya que permite trabajar con todas las plataformas como si fueran una sola (sobre todo desde el punto de vista del administrador), ahorrándose con esto una gran cantidad de tiempo en el desarrollo de sistemas.

Las plataformas en las que actualmente funciona Java son:

- SPARC Solaris (2.3 o posterior)
- Intel x86 Solaris
- Windows 3.x
- Windows NT/95 (Intel x86)
- Macintosh 7.5
- AIX
- HP-UX

- Digital UNIX
- NCR SysV
- Sony NEWS
- Linux

Los visualizadores que actualmente soportan Java son:

- HotJava
- Netscape Navigator
- Internet Explorer

Actualmente existe un grupo de desarrollo de GNU llamado JOLT que está trabajando para portar la máquina virtual de Java y su compilador a otras plataformas como:

- Amiga
- NeXT
- OS/2

Características de Java

A continuación se enumeran algunas características del lenguaje que lo hacen atractivo para su uso en nuestro sistema.

- **Simple**. Debido a que en Java no es necesario manejar apuntadores de forma explícita y a que incluye un recolector automático de basura Java resulta ser un lenguaje sencillo de aprender y adecuado para desarrollar sistemas complejos evitando al desarrollador muchos errores comunes en otros lenguajes, como por ejemplo: que la memoria reservada no sea liberada posteriormente, o sea liberada antes de tiempo, o se trate de liberar más de una vez. Este trabajo lo desempeña de forma óptima el recolector de basura de Java debido a que el recolector de basura es ejecutado como un *thread* (hilo de control) con una prioridad muy baja, siendo imperceptible para el usuario o el programador. Por otra parte, debido a que su sintaxis es muy parecida a la de C o C++, el código escrito en este lenguaje se puede entender fácilmente. Aunque hace uso de la sintaxis de C++ Java elimina muchos de las opciones que están permitidas en C++, como la sobrecarga de operadores, y conversiones automáticas entre tipos.

- **Orientado a objetos**. Java como todo lenguaje orientado a objetos proporciona facilidades como el manejo de la herencia, el polimorfismo y el encapsulamiento. Esto ayuda en gran medida al programador, si anteriormente ha realizado un análisis y un diseño orientado a objetos; ya que el lenguaje le permitirá realizar la implementación de una forma más sencilla y directa. Otra parte importante es que en Java siempre tenemos que utilizar la metodología orientada a objetos (no hay forma de programar en forma estructurada), esto contrasta notablemente con otros lenguajes como C++ que si lo permite. Como Java utiliza únicamente herencia simple, le

ahorra al programador todos los problemas que se pueden provocar por el uso de herencia múltiple en sus programas, por ejemplo la resolución de métodos con el mismo nombre pero que provienen de diferentes ramas de la jerarquía de clases. Al mismo tiempo de que proporciona otras formas sencillas de implementar sistemas que anteriormente requerían del uso de la herencia múltiple (interfaces).

- **Distribuido**. Java está diseñado para desarrollar aplicaciones que trabajen en redes. Java soporta diferentes niveles de conectividad de redes a través de un paquete conocido como ***Java.net***. Esto implica poder trabajar con protocolos de comunicación tanto de las primeras capas del modelo *OSI* (*TCP/IP*), como con protocolos de las capas más altas (*HTTP* y *FTP*). Además, los proveedores tienen planeado incluir dentro de las bibliotecas estándar de Java, un conjunto de clases para el manejo de mensajes remotos (algo parecido a las *RPC* pero con métodos), y la creación de objetos que puedan viajar a través de la red y ejecutarse.

- **Interpretado**. El compilador de Java genera *bytecodes*, en vez de código de máquina nativo. Para ejecutar una aplicación es necesario suministrar este *bytecode* a la máquina virtual de Java para que los pueda interpretar. Esto implica una cierta degradación en el tiempo de respuesta de la aplicación. No obstante, actualmente existen implementaciones de una tecnología conocida como *just-in-time*. Esta tecnología permite al momento de ejecución generar código de la máquina nativa a partir de los *bytecodes*, esto repercute en un incremento notable en la velocidad de ejecución de los programas. Además, con el surgimiento de arquitecturas de hardware que ejecutan directamente el *bytecode* se pueden obtener velocidades de ejecución óptimas para las aplicaciones. Además de todo esto el hecho de ser interpretado nos proporciona todas las ventajas para depurar y ejecutar los programas de forma incremental, sin tener que estar sujetos al antiguo ciclo de: escribir-compilar-ligar-ejecutar-depurar-escribir

- **Robusto**. Java es un lenguaje fuertemente tipificado, lo cual implica que al momento de compilación el compilador de Java es capaz de advertir al usuario de posibles errores semánticos, los cuales por lo regular son difíciles de depurar. Además el lenguaje tiene interconstruida la capacidad de manejar excepciones, esto permite al programador contar con una forma estándar de manejo de posibles errores que se pueden presentar al momento de ejecución, logrando con esto sistemas más estables al momento de ser usados por los usuarios finales. Además, el esquema con el que cuenta Java para el uso de memoria nos garantiza que no existirá forma de dañar o liberar memoria de forma accidental.

- **Seguro**. Debido a que Java está pensado para ser utilizado en redes es muy importante que el propio lenguaje cuente con varias formas de garantizar la integridad, la consistencia y la privacidad de los programas. De hecho Java cuenta con varios mecanismos y niveles para que se puedan desarrollar sistemas y que éstos puedan viajar por la red de forma confiable, como puede ser la inclusión de algoritmos de cifrado mediante el uso de llave

pública. Además, la máquina virtual proporciona todo un mecanismo para verificar y autentificar los *bytecodes* que está ejecutando en un momento dado, evitando con esto ejecutar programas que puedan dañar los sistemas del usuario o que proporcionen información confidencial a otras personas.

- **Arquitectura neutral**. Debido a que el código fuente de Java es traducido por el compilador en *bytecodes*, y este código no es exclusivo de ningún hardware, se puede tener la certeza de que un programa funcionara perfectamente en cualquier plataforma que tenga incluida la máquina virtual de Java. Actualmente la gran mayoría de plataformas cuentan con una implementación de la máquina virtual. El surgimiento de las máquinas Java en hardware no implica ningún retroceso en este sentido; ya que aunque el código será ejecutado de manera más eficiente en estas máquinas, todavía podrá el mismo código ser ejecutado en las demás plataformas, gracias a la máquina virtual. Además, el mismo lenguaje estandariza el tamaño de los tipos de datos primitivos, Por ejemplo, un entero siempre ocupara 32 bits. Las bibliotecas que forman parte del sistema definen interfaces portables. Por ejemplo, hay una clase abstracta ***Window*** e implementaciones de ésta para *Unix*, *Windows* y *Macintosh*.

- ***Multithreaded***. Java tiene la capacidad de crear múltiples *threads*, lo cual permite la ejecución en forma concurrente de objetos. Para que esta ejecución se realice adecuadamente Java cuenta con un sofisticado conjunto de primitivas de sincronización que están basadas en el ampliamente usado paradigma de monitor y variable de condición introducido por C.A.R.Hoare. Mucho del estilo de ésta integración viene del sistema Cedar/Mesa de Xerox. Otras ventajas son las de poder obtener una mejor interacción con el sistema y que éste tenga una respuesta casi en tiempo real

En este momento la compañía SUN acaba de liberar la versión 1.1 de su *JDK* (Java Development Kit) que incluye las siguientes características:

1. **Archivos JAR**. JAR es un formato de archivo independiente de plataforma que contiene varios archivos dentro de uno solo. Múltiples *applets* y los componentes que requieren (archivos .class, imagenes y sonidos) pueden ser incluidos en un archivo JAR y subsecuentemente ser cargados en el *visualizador* en una sola transacción *HTTP*, mejorando significativamente el tiempo de transferencia. El formato JAR también acepta compresión, lo cual reduce el tamaño del archivo y disminuye el tiempo de transferencia. Además, el autor de la aplicación puede firmar digitalmente el archivo para autentificar su origen.

2. **Internacionalización**. Permite el desarrollo de *applets* localizables. Es decir, incluye mejoras para desplegar caracteres *UNICODE* mediante un mecanismo local, es sensible a la fecha, hora, tiempo y horario local, e incluye un convertidor del conjunto de caracteres, etc.

3. **Seguridad.** El *JDK* incluye *APIs* para firmas digitales, manejo de llaves, listas de control de acceso, e incluye la posibilidad de firmar clases y otros datos.

4. **Mejoras al AWT.** Está conformado por una vasta infraestructura para el desarrollo de GUIs a gran escala, además, contiene APIs para imprimir, para hacer uso del *clipboard*, manejando diferentes tipos de cursores por componente, un modelo de eventos basado en delegación, mejoras a los gráficos y las imágenes, y un soporte más flexible para el uso de fonts.

5. **Mejoras a la interfaz de red.** Lo cual incluye el soporte para seleccionar sockets estilo *BSD*. Las clases ***Socket*** y ***ServerSocket*** pueden ser extendidas a través de herencia. Se agregaron nuevas subclases a la clase ***SocketException*** para tener mayor granularidad en el reporte y manejo de errores.

6. **Invocación de métodos remotos (RMI).** Lo cual permite a los objetos de Java invocar métodos de objetos que se encuentran en otras máquinas virtuales, inclusive en otras computadoras. Referencias a tales objetos remotos pueden ser pasados como parámetros en llamadas RMI.

7. **Serialización de Objetos.** Extiende las clases base de entrada/salida de Java para dar soporte a objetos. Permitiendo la codificación de objetos y la reconstrucción de estos provenientes de un flujo de bytes. La serialización es utilizada para la persistencia de los objetos o para comunicación vía sockets o RMI. La codificación de los objetos protege los datos privados y temporales.

8. **Clases *Byte*, *Short* y *Void*.** Extiende la biblioteca de clases para incluir clases que permiten manipular a los tipos primitivos como objetos.

9. **JDBCTM - Java Database Connectivity.** JDBC es una interfaz de acceso estándar para bases de datos SQL. Éste también proporciona una base común sobre la cual pueden ser construidas herramientas e interfaces de alto nivel. Incluye un bridge ODBC. El bridge o puente es una biblioteca que implementa el JDBC (Conectividad para Bases de Datos de Java) en términos del *API* estándar para ODBC.

10. **Nueva interfaz para métodos nativos Java.** Una interfaz de programación estándar para escribir métodos nativos. El principal objetivo es obtener compatibilidad a nivel binario de bibliotecas de métodos nativos a través de todas las implementaciones de la máquina virtual en una plataforma específica.

11. **Documentación para la interfaz del compilador JIT.** Esta documentación es para desarrolladores de herramientas quienes

están escribiendo generadores de código nativo u otras utilerías que corren dentro de la máquina virtual de Java.

12. **Clases anidadas**. Provee una sintaxis simple para la creación de clases *Adapter*. Una clase adapter es una clase que implementa una interfaz (o clase) requerida por un *API*, y delega el flujo de control de regreso a un objeto "principal".

13. **Mejoras en el rendimiento**. Mejoras en el ciclo interno del intérprete ahora escrito en ensamblador sobre Win32 y Solaris/Sparc. Las Clases peer del *AWT* han sido reescritas para Win32. Y un uso opcional de *threads* nativos sobre Solaris.

Programación Orientada a Objetos

Por otra parte, la programación orientada a objetos por sí misma ha mostrado ser una buena herramienta para la construcción de sistemas complejos y reusables. Esta tecnología provee al programador de técnicas y guías para lograr estos objetivos de una forma clara y consistente. Además, actualmente se están consolidando estándares para la utilización de esta tecnología con sistemas ya existentes o con nuevos dentro de las áreas de procesamiento distribuido, redes, bases de datos, etc., como por ejemplo *CORBA*.

Con lo anterior se cubren las características principales del lenguaje Java dando por concluido este punto, pudiendo proceder en el siguiente capítulo a exponer las características de la segunda tecnología propuesta, es decir Agentes.

Capítulo 3

Agentes.

En este capítulo se trata el surgimiento y evolución del concepto de agente. Se muestra una definición de lo que consideramos un agente y se hace una clasificación de los diferentes tipos de agentes. También se mencionan las instituciones que están investigando y haciendo desarrollos en esta área y algunas de las áreas donde se hace uso de esta tecnología.

Antecedentes

El software de agentes surgió como parte de la evolución de los sistemas multiagentes (MAS), el cual forma una de las tres amplias áreas que caen bajo el área de la Inteligencia Artificial Distribuida (DAI), las otras dos áreas son la Solución a Problemas Distribuidos (DPS) y la Inteligencia Artificial Paralela (PAI). Por tanto, como los sistemas multiagentes, el software de agentes hereda muchas de las motivaciones, objetivos y beneficios potenciales de la DAI. Por ejemplo, gracias al cómputo distribuido, el software de agentes hereda modularidad, velocidad (debido al paralelismo) y confiabilidad (debido a la redundancia). También hereda aspectos de la Inteligencia Artificial tales como operaciones a nivel del conocimiento, facilidad de mantenimiento, reusabilidad e independencia de la plataforma.

El concepto de agente puede remontarse a los primeros días de investigación de la DAI en los 70s debido al modelo de actor de Carl Hewitt. En este modelo, Hewitt propuso el concepto de un objeto autocontenido, interactivo y ejecutado concurrentemente al cual llamó "actor". Este objeto tenía un estado interno encapsulado y podía responder a mensajes de otros objetos similares. Desde el punto de vista de Hewitt "un actor es un agente computacional el cual tiene una dirección de correo y un comportamiento. Los actores se comunican al pasar mensajes y llevar a cabo sus acciones concurrentemente".

La tecnología de software de agentes ha crecido considerablemente en los últimos años y se ha aplicado para resolver algunos de los problemas derivados del acelerado crecimiento de la red Internet y del servicio *WWW*. Actualmente, y casi principalmente, los agentes se utilizan para la creación de sistemas de búsqueda de información sobre Internet y también como asistentes personales. Sin embargo, el término agente es mal usado en el mercado del software debido a que los desarrolladores de la mayoría de las aplicaciones actuales (particularmente aquéllas para el World Wide Web) hacen creer a sus usuarios que sus aplicaciones poseen cierta inteligencia que permite que el software ejecute tareas de manera automática, desafortunadamente eso no siempre es cierto.

Definición

Podemos definir un agente como alguien o algo que actúa en representación de otra parte, con el propósito de ejecutar acciones específicas en beneficio de la parte representada. Un agente de software es un programa que ejecuta tareas para su usuario dentro de un ambiente de cómputo. Técnicamente hablando, la mayoría de las aplicaciones de software podrían ser definidas como agentes, sin embargo, las aplicaciones que hacen uso de agentes son diferenciadas de otras por sus características de movilidad, autonomía, y capacidad de interactuar independientemente de la presencia del usuario. Además, cuando se introduce el elemento adicional de inteligencia a un agente, se debe incluir la capacidad para razonamiento adaptivo. Esto implica la capacidad que deberá tener el agente para procesar información desde ambientes externos (tales como redes, bases de datos, e Internet) dado un conjunto de conocimientos, actitudes, y creencias del usuario, las cuales son entendidas por el agente.

Características

La teoría de agentes ha conformado una de las áreas con mayor crecimiento en Internet debido a:

- La inquietud existente entre los investigadores en llevar a cabo en forma más eficiente la adquisición y recuperación de información distribuida remotamente en una gran cantidad de servidores localizados en algunos de los múltiples países interconectados a ella.

- El interés de los proveedores de equipos y software para ganar un mercado que les permitirá vender y rentar servicios tanto de búsqueda de información como de uso de paquetes, juegos y sistemas.

Algunas características que hacen diferentes a los agentes de otros tipos de software de aplicación son las siguientes:

Autonomía. El agente debe tener la capacidad de tomar acciones que lo conduzcan a completar algunas tareas u objetivos, sin necesidad de que el usuario final se lo indique explícitamente. Debe existir un elemento de independencia para el agente, parecido a la manera en que trabajan los agentes humanos, ellos toman el problema, intereses, deseos, etc. como entrada y continúan por sí mismos para efectuar las tareas esperadas. Los agentes de software deben tener "control" sobre su "estado y comportamiento interno". Esto implica que un agente deba tener acceso a una red y tener la movilidad a través de ésta. El agente es motivado ya sea por su usuario o por el propio ambiente para llevar a cabo su misión, por lo tanto busca oportunidades para progresar en sus objetivos. En el contexto del World Wide Web, la autonomía del agente se traduce como la capacidad para operar en el dominio de Internet, mientras que el usuario está desconectado o fuera de la interacción del WWW.

Los agentes entienden su misión y proceden a llevarla a cabo, viajando en el WWW para encontrar recursos que lo ayuden a cumplir con sus tareas.

Capacidad de comunicación. Los agentes deben accesar a información de fuentes de una tercera parte acerca del "estado" actual del ambiente externo mientras llevan a cabo sus objetivos. Esto requiere de una liga de comunicación entre los agentes y los repositorios de información, los cuales pueden ser otros agentes o dispositivos retenedores de la información. Dicha comunicación puede ser llevada a cabo en forma de una solicitud, a través de un conjunto de preguntas y respuestas ya establecidas, concisas y simples, o bien podría ser una comunicación compleja con respuestas variables. La comunicación podría ser hecha a un alto nivel implicando un diálogo verdadero, opuesto al manejo de protocolos. La comunicación entre agentes se puede describir en términos de una negociación. Los agentes hacen ver sus intenciones y objetivos, eventualmente llegando a algún acuerdo o "contrato". Para los agentes inteligentes basados en el WWW, la capacidad para la comunicación apropiada es de suma importancia para que logren realizar sus tareas exitosamente.

Capacidad de cooperación. Una extensión natural del atributo de comunicación es la cooperación. Los agentes deben tener un espíritu colaborativo para existir y tener éxito en lo que se conoce como sistemas orientados a agentes, en donde los agentes trabajan en conjunto para llevar a cabo tareas complejas pero mutuamente benéficas.

Capacidad para razonar. La capacidad para razonar es uno los aspectos claves de la inteligencia que distingue a los agentes inteligentes de otros tipos de agentes. El razonar implica que un agente pueda poseer la capacidad para inferir y extrapolar basándose en las experiencias y los conocimientos que ya posee. A continuación describimos tres tipos de escenarios de razonamiento:

- Basado en reglas, donde los agentes usan un conjunto de pre-condiciones dadas por el usuario para evaluar condiciones en el ambiente externo,

- Basado en el conocimiento, donde los agentes son provistos de conjuntos grandes de datos relacionados con otros escenarios y de resultados de acciones, desde las cuales ellos deducen sus futuros movimientos, y

- Basado en la evolución artificial, donde los agentes "producen" nuevas generaciones de agentes que poseen mejores capacidades de razonamiento.

El razonamiento es altamente dependiente del grado en el cual los agentes tengan la capacidad de demostrar, atributos tales como: emociones, creencias, e intenciones, como los seres humanos.

Comportamiento adaptivo. Para que los agentes puedan mantener sus capacidades de autonomía y razonamiento, el agente debe poseer algún

mecanismo para determinar el estado corriente de su dominio externo, el cual se define como la extensión del ambiente contenido en el alcance del agente, e incorporando éste dentro de sus "decisiones" acerca de sus futuras acciones. Los agentes deberían tener la capacidad de examinar el ambiente externo y retomar las acciones previas que hayan sido exitosas bajo condiciones similares y adaptar sus acciones para mejorar la probabilidad de éxito para llevar a cabo sus objetivos.

Confiabilidad. Para la aceptación de los agentes por parte de su usuario es esencial que exista un fuerte sentido de confianza para que el agente pueda representarlo. Esto es igualmente cierto para los agentes en el World Wide Web. Los agentes deben demostrar "veracidad" y "benevolencia". En otras palabras, el usuario debe estar altamente seguro de que el agente actuará y reportará confiablemente, y actuará por el bien del usuario.

Clasificación

Existen diferentes puntos de vista desde los cuales se puede hacer una clasificación del software de agentes.

Primero, los agentes pueden ser clasificados por su movilidad, es decir, por su habilidad para moverse a través de una red. Esto da lugar a que los agentes sean considerados como estáticos o móviles.

Segundo, los agentes pueden ser clasificados además como deliberativos o reactivos. Los agentes deliberativos derivan del paradigma de pensamiento deliberativo: los agentes poseen un modelo de razonamiento simbólico interno y se mantienen en planeación y negociación para llevar a cabo la coordinación con otros agentes. Los agentes reactivos, por el contrario, no tienen un modelo simbólico interno de su ambiente, y actúan usando un tipo de comportamiento respuesta/estímulo al responder al estado presente del ambiente en el cual están embebidos.

Tercero, los agentes pueden ser clasificados también de acuerdo a diferentes atributos ideales y primarios que todos deberían exhibir: autonomía, aprendizaje y cooperación.

La autonomía se refiere al principio de que los agentes pueden operar por sí mismos sin necesidad del apoyo humano. Por tanto los agentes tienen estados internos y objetivos, y actúan de acuerdo a sus objetivos en beneficio de sus usuarios. Un elemento clave de su autonomía es su proactividad, es decir, su habilidad para tomar la iniciativa en vez de actuar simplemente en respuesta a su ambiente. La cooperación con otros agentes es muy importante: es la razón de tener múltiples agentes en vez de tener uno solo. Para cooperar los agentes necesitan poseer una habilidad social, es decir, la habilidad para interactuar con otros y posiblemente con humanos, a través de algún lenguaje de comunicación. Finalmente, para que un sistema de agentes sea realmente inteligente, los agentes deben aprender como reaccionar y/o como interactuar con su ambiente

externo, pues el atributo clave de cualquier ser inteligente es su capacidad para aprender.

De acuerdo a estas tres características mínimas se pueden derivar cuatro tipos de agentes: agentes colaborativos, agentes colaborativos que aprenden, agentes de interfaz, y agentes inteligentes confiables.

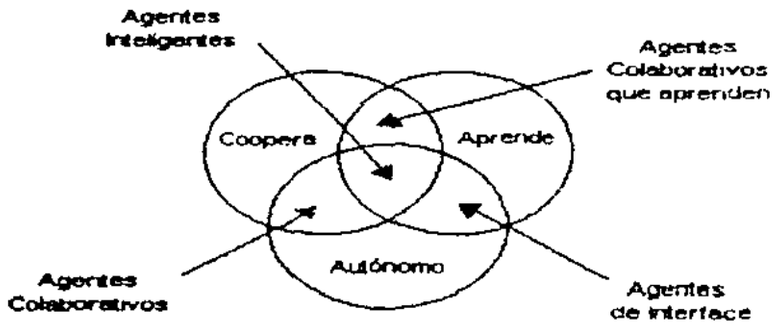


Figura 5 Parte de una tipología de agentes

Las distinciones anteriores no son definitivas. Por ejemplo, con agentes colaborativos, existe más énfasis sobre cooperación y autonomía que sobre aprendizaje, pero eso no implica que los agentes colaborativos nunca aprendan. Por otro lado, para agentes de interfaz, se hace más énfasis sobre autonomía y aprendizaje que sobre cooperación. Cualquier cosa que se encuentra fuera de las áreas de intersección no puede ser considerada como agente. Por ejemplo, la mayoría de los sistemas expertos son autónomos pero, generalmente, no cooperan o aprenden. Idealmente, los agentes deberían hacer las tres cosas bien de igual manera, pero esto es más que nada la aspiración de sus creadores en vez de la realidad.

En principio, al combinar las dos construcciones, es decir, estático/móvil y reactivo/deliverativo, en conjunción con los tipos de agentes identificados (agentes colaborativos, agentes de interfaz, etc.), podríamos tener agentes colaborativos deliberativos estáticos, agentes colaborativos reactivos móviles, agentes de interfaz deliberativos estáticos, agentes de interfaz reactivos móviles, etc. Lo que hace tener 16 diferentes categorías las cuales pueden ser necesarias para posteriormente clasificar a los agentes existentes.

Cuarto, los agentes pueden algunas veces ser clasificados por el papel que desempeñan (preferiblemente si el papel es importante), por ejemplo los agentes de información del World Wide Web. Esta categoría de agentes usualmente explota los medios de búsqueda tales como *WebCrawlers*, *Lycos* y *Spiders*. Esencialmente, ayudan a administrar la gran cantidad de información de las redes como Internet. Los agentes de información pueden ser a su vez estáticos, móviles o deliberativos o bien pueden ser tipificados además con base en otros

papeles menos importantes tales como los de: agentes de reportes, agentes de presentación, agentes de análisis y diseño, agentes de prueba, agentes de empacadores y agentes de ayuda.

Quinto, se puede incluir la categoría de agentes híbridos la cual combina dos o más filosofías de agentes en un agente.

En esencia, los agentes existen en un espacio multidimensional, sin embargo existe una lista de siete tipos de agentes, la cual cubre a la mayor parte de los tipos de agentes sobre los cuales actualmente se ha hecho investigación, a saber:

- Agentes Colaborativos.
- Agentes de Interfaz.
- Agentes Móviles.
- Agentes de Información.
- Agentes Reactivos.
- Agentes Híbridos.
- Agentes Inteligentes.

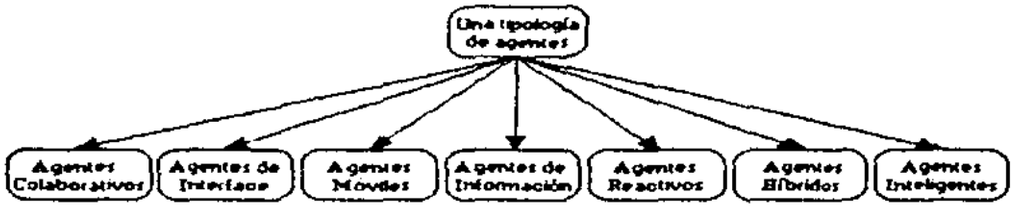


Figura 6 Una clasificación de agentes

Agentes Colaborativos.

Los agentes colaborativos dan énfasis a la autonomía y cooperación (con otros agentes) para llevar a cabo tareas para sus propietarios. Ellos pueden aprender, pero este aspecto no es generalmente importante dentro de su operación. Para tener un conjunto de agentes colaborativos coordinados, los agentes tienen que negociar para alcanzar acuerdos aceptables en algunos aspectos. Las características clave de estos agentes incluyen autonomía, habilidad social, responsabilidad y proactividad. Por tanto, ellos son (o deberían ser) capaces de actuar racionalmente y autónomamente en ambientes abiertos y de múltiples agentes restringidos en tiempo. Tender a ser estáticos. Poder ser benevolentes, racionales, confiables, o bien alguna combinación de estas o ninguna de ellas. Típicamente la mayoría de los agentes colaborativos implementados hasta la fecha no ejecutan aprendizaje complejo.

Agentes de Interfaz.

Los agentes de interfaz dan énfasis a la autonomía y al aprendizaje para ejecutar las tareas de sus usuarios. El aspecto clave en el que se basa este tipo de agentes es en el de un asistente personal que está colaborando con el usuario en el mismo ambiente de trabajo. Se debe notar el énfasis y hacer distinción entre la colaboración con el usuario y la colaboración con otros agentes como es el caso de los agentes colaborativos. Colaborar con el usuario podría no requerir un lenguaje de comunicación de agentes como el requerido cuando se colabora con otros agentes. Esencialmente, los agentes de interfaz apoyan y ofrecen asistencia a un usuario aprendiendo a usar una aplicación en particular, tal como una hoja de cálculo o un sistema operativo. El agente observa y monitorea las acciones tomadas por el usuario en la interfaz, aprende nuevos atajos, y sugiere mejores formas de hacer la tarea. Además el agente del usuario actúa como un asistente personal autónomo el cual coopera con él para llevar a cabo una tarea en la aplicación. Los agentes de interfaz aprenden para asistir mejor a su usuario de cuatro formas diferentes:

- Al observar e imitar al usuario (es decir, emula al usuario).
- A través de recibir retroalimentación positiva y negativa del usuario (aprende del usuario).
- Al recibir instrucciones explícitas del usuario (aprende del usuario).
- Al pedir sugerencias a otros agentes (es decir, aprende de otros agentes).

Su cooperación con otros agentes, si es que ésta existe, está limitada a solicitar sugerencias, y no a hacer contratos como es el caso con los agentes colaborativos.

Agentes móviles.

Los agentes móviles son procesos de software capaces de transportarse en WANs, interactuar con máquinas foráneas, recolectar información en beneficio de su usuario y regresar una vez que ha ejecutado las actividades que tenía a su cargo. Las actividades pueden ser desde una reservación de vuelo hasta la administración de una red de telecomunicaciones. Los agentes móviles son autónomos y cooperan aunque de diferente manera que los agentes colaborativos. Por ejemplo, ellos pueden cooperar o comunicarse con otros agentes al dar a conocer su estado interno y métodos. Al hacer esto, un agente intercambia datos o información con otros agentes sin tener que presentar toda su información.

El concepto de agentes móviles surgió de una revisión acerca de la forma en que las computadoras se habían comunicado hasta finales de los 70's. A continuación se exponen los resultados de dicha revisión, se explica el por qué los agentes móviles ofrecen una mejor solución a los paradigmas de interacción y de comunicación entre computadoras.

La llamada a procedimientos remotos (*RPC*), la cual fue concebida en los 70s, es el principio de organización central de las redes de comunicación entre las computadoras actuales. El paradigma *RPC* visualizaba a la comunicación computadora-a-computadora como una forma de permitirle a una computadora hacer llamadas a procedimientos dentro de otra. Los mensajes que la red transporta pueden ser solicitudes o respuestas a la ejecución de procedimientos. Un mensaje incluye datos que pueden ser argumentos para la ejecución de metodos remotos. La respuesta incluye datos que son sus resultados. El procedimiento por si mismo es interno a la computadora que lo lleva a cabo. Este paradigma de comunicación se muestra en la siguiente figura:

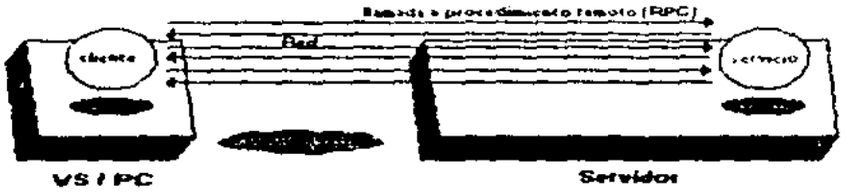


Figura 7 Llamada a procedimiento remoto

Dos computadoras cuya comunicación sigue el paradigma *RPC* acuerdan sobre los efectos de cada procedimiento accesible de manera remota y los tipos de argumentos y resultados. Sus acuerdos constituyen un protocolo. Una computadora cliente con trabajo para un servidor, realiza su labor con una serie de llamadas a procedimientos remotos. Cada llamada implica una solicitud enviada desde el cliente hasta el servidor, y la respuesta es enviada desde el servidor hasta el cliente.

Un ejemplo sencillo nos permite ver el problema de esta propuesta. Para borrar todos los archivos con al menos dos meses de antigüedad en el servidor de archivos, una computadora cliente podría hacer una *RPC* para obtener los nombres y fechas de los archivos del servidor y otra por cada archivo a ser borrado. El análisis que decide cuales archivos son lo suficientemente antiguos para ser borrados es realizado en la computadora cliente. Si ésta decide borrar n archivos, la computadora cliente debe enviar un total de 2(n+1) mensajes. La característica sobresaliente de la llamada a procedimientos remotos es que cada interacción entre la computadora del usuario y el servidor entablan dos actos de comunicación, uno para solicitar que el servidor ejecute un procedimiento, y otro para dar a conocer que el servidor lo hizo.

Una alternativa a la llamada a procedimientos remotos es la programación remota (*RP*), la cual es la clave para la tecnología de agentes. El paradigma *RP* visualiza la comunicación computadora-a-computadora como una forma de permitir a una computadora no sólo a hacer llamadas a procedimientos en otra, sino que también proporciona los procedimientos a ser ejecutados. Cada mensaje que la red transporta consiste de: un procedimiento que la computadora

receptora ejecuta y los datos que funcionan como argumentos para el procedimiento. El procedimiento se comienza a ejecutar en la computadora que lo envía y continua en la computadora receptora. Este paradigma de comunicación es mostrado en la siguiente figura.

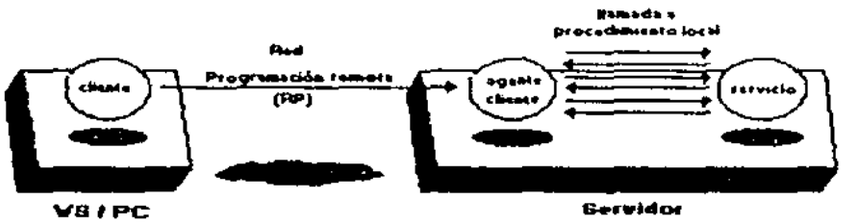


Figura 8 Programación remota

Dos computadoras cuya comunicación se hace de acuerdo al paradigma *RP* acuerdan sobre las instrucciones que serán permitidas en los procedimientos y en los tipos de datos que serán permitidos. Dichos acuerdos constituyen un lenguaje. Éste incluye instrucciones que permiten a los procedimientos tomar decisiones, examinar y modificar su estado, además de poder llamar a otros procedimientos en la computadora receptora. Tales llamadas a procedimientos son hechas de manera local en vez de en forma remota. El procedimiento y su estado son conocidos como un agente móvil para enfatizar que representan a la computadora emisora aún cuando el procedimiento y su estado se encuentren en la computadora receptora.

Una computadora cliente con trabajo para un servidor (por ejemplo borrar archivos con al menos dos meses de antigüedad), envía un agente cuyo procedimiento hace las solicitudes requeridas dentro del servidor basado en su estado, supongamos "archivos con dos meses de antigüedad". El procedimiento requiere sólo del mensaje que transporta al agente entre las computadoras. El agente, no la computadora cliente, coordina el trabajo, decidiendo cuales archivos deberán ser borrados. La característica sobresaliente de la programación remota es que la computadora cliente y el servidor pueden interactuar sin utilizar la red una vez que la red ha transportado al agente entre ellas.

La programación remota tiene dos ventajas importantes sobre las llamadas a procedimientos remotos: una táctica y la otra estratégica.

La ventaja táctica de la programación remota es la ejecución. Cuando una computadora cliente tiene trabajo para que un servidor lo haga, en vez de direccionar comandos a través de la red, ésta envía un agente hacia el servidor y direcciona el trabajo localmente en vez de hacerlo en forma remota. De esa manera la red transporta pocos mensajes. Entre más trabajo se deba realizar, la programación remota evita el envío de más mensajes.

La ventaja estratégica de la programación remota es la capacidad para hacer modificaciones. Los agentes permiten a los fabricantes de software para máquinas cliente extender la funcionalidad ofrecida por los fabricantes de software para máquinas servidor. Regresando al ejemplo de los archivos , si el servidor provee un procedimiento para listar los archivos de un cliente y otro para borrar un archivo por nombre, un cliente puede agregar a ese repertorio un procedimiento que borre todos los archivos con un tiempo de existencia específico. El nuevo procedimiento, el cual toma la forma de un agente, personaliza el servidor para ese cliente. El paradigma de la programación remota cambia no sólo la división de la labor entre los fabricantes de software, sino también la facilidad de instalación del software que ellos producen.

A diferencia de las aplicaciones ejecutables tradicionales, las aplicaciones de comunicación que surgen de Internet tienen componentes que deben residir en los servidores. Los componentes en el servidor de una aplicación basada en *RPC* deben ser instalados por el usuario. Por otro lado, los componentes del servidor de una aplicación basada en RP son dinámicamente instalados por la aplicación misma. Cada uno es un agente. La ventaja de la programación remota es significativa en la red de una empresa, pero es más importante en una red pública cuyos servidores son propiedad de los proveedores de servicios públicos en el World Wide Web. Introducir una nueva aplicación basada en *RPC* requiere tomar una decisión de negocios por parte del proveedor de servicios. Para una aplicación basada en RP lo único que se requiere es que el usuario tome la decisión. La programación remota convierte una red pública en una plataforma; ya que integra de una forma casi transparente cada uno de sus elementos y los presenta como uno sólo.

Agentes de Información.

Los agentes de información han surgido debido a la gran demanda de herramientas para ayudarnos a manejar el explosivo crecimiento de información que estamos experimentando actualmente. Los agentes de información ejecutan el papel de administración, manipulación o selección de información de muchas fuentes de sistemas distribuidos. Este tipo de agentes están definidos por lo que hacen, a diferencia de los agentes colaborativos o de interfaz los cuales están definidos por lo que son (es decir, sus atributos)

Agentes reactivos.

Los agentes reactivos representan a una categoría especial de agentes los cuales no poseen un modelo simbólico interno de su ambiente, en vez de ello actúan o responden al estado presente del ambiente en el cual están embebidos. Un punto importante a notar con los agentes reactivos no son sus lenguajes, teorías o arquitecturas, sino el hecho de que los agentes son relativamente simples e interactúan con otros agentes de manera básica. Sin embargo, patrones complejos de comportamiento emergen de estas interacciones cuando la unión de agentes es visualizada globalmente.

Existen tres ideas claves que soportan a los agentes reactivos. Primero, la funcionalidad emergente, es decir, la dinámica de la interacción que conduce a la complejidad emergente. Por tanto, no existe una especificación a priori (o plan) del comportamiento inicial del agente. Segundo, la descomposición de tareas: un agente reactivo es visto como una colección de módulos los cuales operan autónomamente y son responsables de tareas específicas (por ejemplo sensar, controlar un motor, calcular, etc.). La comunicación entre los módulos es minimizada y de una naturaleza de bajo nivel. No existe un modelo global contenido en este tipo de agentes, por lo tanto el comportamiento global tiene que emerger. Tercero, los agentes reactivos tienden a operar sobre representaciones las cuales son parecidas a datos sensados sin procesar, a diferencia de la representación simbólica que abunda en otros tipos de agentes.

Agentes Híbridos.

Son aquéllos cuya constitución es una combinación de dos o más tipos de agentes dentro de un sólo agente. Estas tipos incluyen un tipo móvil, un tipo de agente de interfaz, una tipo de agente colaborativo, etc.

Funcionamiento

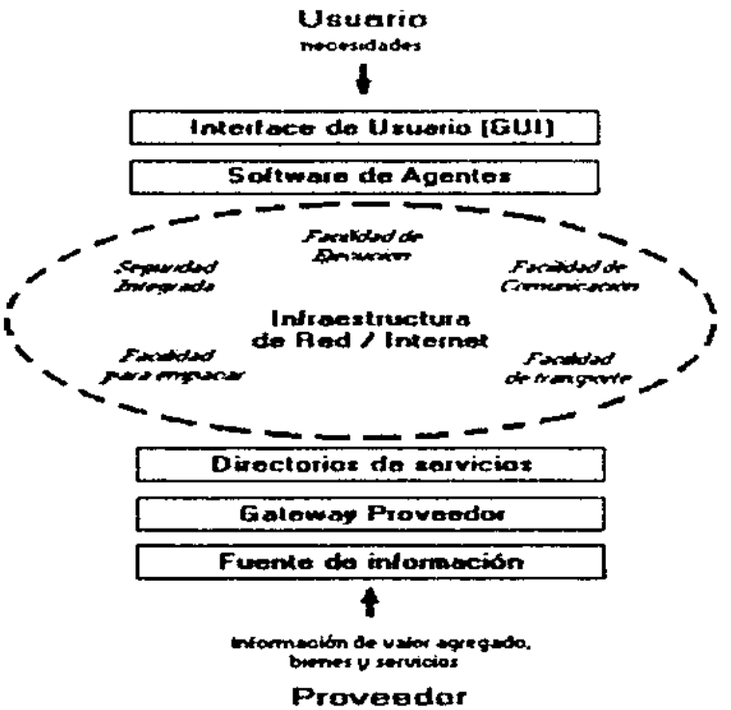


Figura 9 Modelo básico de trabajo de un agente

La figura anterior muestra el modelo básico de trabajo con un agente.

Un usuario para realizar una tarea utiliza una interfaz gráfica (GUI) la cual accesa a una aplicación de software. El agente viaja a través de Internet u otra infraestructura de red o una intranet para alcanzar a los *gateways* proveedores, asistidos por los directorios de servicio para finalmente permitir al agente accesar a la fuente de información en bases de datos u otros medios de almacenamiento, en donde probablemente se encuentra la información solicitada por el usuario. El agente completa cualquier transacción necesaria, y devuelve una respuesta al usuario.

El modelo de transacciones entre agentes muestra la infraestructura de red en el modelo de agente como una elipse nebulosa. A continuación se describe esa infraestructura de red en más detalle. La infraestructura aplicada a los agentes incluye:

1. **Facilidad de ejecución**. Ésta es una variable dependiente del hardware y del software necesario para ejecutar los programas del agente en el ambiente del agente. El hardware incluye cualquier tipo de computadora personal, workstation, o mainframe. Debido a la proliferación de compiladores para lenguajes de programación de agentes, sólo es necesario considerar la velocidad y compatibilidad con el hardware y software con los que se cuenta dentro de la organización para determinar cuál plataforma es la adecuada en la implementación de un sistema de agentes,.

El lenguaje de programación para software de agentes determina el rango de funcionalidad posible en una aplicación de agentes. Hasta el momento, ésta es un área de intenso estudio e investigación por parte de instituciones académicas y comerciales. Dado que las aplicaciones de agentes difieren enormemente, y dado que el campo de ésta tecnología aún está evolucionando, los expertos rara vez se ponen de acuerdo sobre el lenguaje de programación apropiado para el desarrollo de aplicaciones con agentes. Generalmente, las opciones caen dentro de tres categorías: lenguajes de programación general y lenguajes para scripts, lenguajes de código móvil de propósito general, y finalmente lenguajes de código móvil escritos específicamente para aplicaciones de agentes.

a) Lenguajes de programación de propósito general y lenguajes para scripts.

Los lenguajes de programación de propósito general favorecidos para escritura de aplicaciones de agentes debido a su eficiencia incluyen a los lenguajes C y C++, y al lenguaje *Lisp*, el cual es favorecido por su flexible estructura simbólica. *Lisp* es comúnmente utilizado en aplicaciones de inteligencia artificial. Muchas de las aplicaciones sencillas, las cuales son consideradas aplicaciones inteligentes, son en su mayor parte escritas con lenguajes para el desarrollo de *scripts* como Structured Query Language (SQL), comúnmente utilizado para

consultas a bases de datos, o como *Perl*, comúnmente utilizado para el desarrollo de scripts de comandos para el *WWW*.

b) Lenguajes de código móvil de propósito general.

El contar con un lenguaje que produce código móvil añade al propio código la flexibilidad necesaria para ser transmitido a través de la red y ejecutado en otro punto de ésta. Actualmente existen diversos sistemas de código móvil. Java es considerado un lenguaje similar a C++, pero escrito con características extras de movilidad y seguridad para su código. Su aplicación de demostración, HotJava, está convirtiéndose en el estándar para código móvil en el World Wide Web, debido a su integración al visualizador de *WWW* de Netscape.

Safe-Tcl un poco más antiguo que Java es otro de los lenguajes de código móvil pero con menor rendimiento. Otros lenguajes de código móvil, cada uno con sus seguidores devotos, son *Python*, scheme48, Guile, Obliq, Logicware, ScriptX, y Phantom.

c) Lenguajes de código móvil para aplicaciones de agentes inteligentes.

Aunque los lenguajes de código móvil mencionados proveen la funcionalidad necesaria para desarrollar aplicaciones con agentes inteligentes móviles, no fueron desarrollados específicamente para esta clase de aplicaciones. Algunas instituciones académicas y comerciales han desarrollado lenguajes de programación de agentes diseñados para explotar las aplicaciones de los agentes móviles.

El profesor Yoav Shoham de la Universidad de Stanford ha propuesto un "nuevo paradigma, basado en la visión social de la computación". Su concepto de Programacion Orientada a Agentes (*AOP*) ha conducido al desarrollo de AGENT0, un lenguaje de programación para agentes. Este lenguaje a la fecha no está completamente desarrollado pero está disponible en Internet.

IBM se encuentra en el proceso de modificar su exitoso lenguaje de programación *REXX*, bien conocido por su facilidad de uso y sus capacidades en el desarrollo de prototipos, para destinarlo a la tecnología emergente del cómputo basado en agentes. IBM espera que el nuevo lenguaje Object-*REXX* llegue a ser el estándar para la programación de agentes.

Hasta ahora, el lenguaje de programación de agentes que ha recibido la mayor atención es *Telescript* de General Magic. *Telescript* parece ser el estándar emergente para la comunicación basada en agentes. Éste es un lenguaje de programación remota, orientado a objetos, una plataforma que permite la creación de aplicaciones de red activas y distribuidas. *Telescript* permite a los programadores escribir aplicaciones para

consultas a bases de datos, o como *Perl*, comúnmente utilizado para el desarrollo de scripts de comandos para el *WWW*.

b) Lenguajes de código móvil de propósito general.

El contar con un lenguaje que produce código móvil añade al propio código la flexibilidad necesaria para ser transmitido a través de la red y ejecutado en otro punto de ésta. Actualmente existen diversos sistemas de código móvil. Java es considerado un lenguaje similar a C++, pero escrito con características extras de movilidad y seguridad para su código. Su aplicación de demostración, HotJava, está convirtiéndose en el estándar para código móvil en el World Wide Web, debido a su integración al visualizador de *WWW* de Netscape.

Safe-Tcl un poco más antiguo que Java es otro de los lenguajes de código móvil pero con menor rendimiento. Otros lenguajes de código móvil, cada uno con sus seguidores devotos, son *Python*, scheme48, Guile, Obliq, Logicware, ScriptX, y Phantom.

c) Lenguajes de código móvil para aplicaciones de agentes inteligentes.

Aunque los lenguajes de código móvil mencionados proveen la funcionalidad necesaria para desarrollar aplicaciones con agentes inteligentes móviles, no fueron desarrollados especificamente para esta clase de aplicaciones. Algunas instituciones académicas y comerciales han desarrollado lenguajes de programación de agentes diseñados para explotar las aplicaciones de los agentes móviles.

El profesor Yoav Shoham de la Universidad de Stanford ha propuesto un "nuevo paradigma, basado en la visión social de la computación". Su concepto de Programación Orientada a Agentes (*AOP*) ha conducido al desarrollo de AGENT0, un lenguaje de programación para agentes. Este lenguaje a la fecha no está completamente desarrollado pero está disponible en Internet.

IBM se encuentra en el proceso de modificar su exitoso lenguaje de programación *REXX*, bien conocido por su facilidad de uso y sus capacidades en el desarrollo de prototipos, para destinarlo a la tecnología emergente del cómputo basado en agentes. IBM espera que el nuevo lenguaje Object-*REXX* llegue a ser el estándar para la programación de agentes.

Hasta ahora, el lenguaje de programación de agentes que ha recibido la mayor atención es *Telescript* de General Magic. *Telescript* parece ser el estándar emergente para la comunicación basada en agentes. Este es un lenguaje de programación remota, orientado a objetos, una plataforma que permite la creación de aplicaciones de red activas y distribuidas. *Telescript* permite a los programadores escribir aplicaciones para

facilitar a los agentes la navegación a través de una variedad de servicios y de diferentes sistemas de cómputo para ejecutar tareas para usuarios. Además de proveer el lenguaje de programación para la ejecución del ambiente, también cubre aspectos de comunicación y seguridad. Su tecnología es considerada superior a la de sus competidores para aplicaciones de agentes, aunque la tecnología no es tan madura como muchos de los lenguajes de propósito general y lenguajes de código móvil. *Telescript* se encuentra bien situado en aplicaciones de comercio electrónico.

2. **Facilidad de comunicación**. Esencialmente la facilidad de comunicación tiene que ver con lo que el agente está comunicando. La comunicación es el proceso mediante el cual la información es intercambiada en una transacción de agentes. Los mensajes comunicados tienen contenido y contexto. El contenido se refiere a los datos codificados en el mensaje; el contexto determina los cambios de datos a la información.

La facilidad de comunicación debe soportar comunicación síncrona y asíncrona, y debe ser capaz de comunicar simultáneamente a múltiples agentes diferentes.

La Arquitectura de Agente Unificada, propuesta por Marc Belgrave en la Universidad McGill en Montreal, Canadá, define la facilidad de comunicación como un conjunto de protocolos estándares que permiten que la comunicación se lleve a cabo. A la fecha ninguna facilidad de comunicación estándar ha emergido, sin embargo el ARPA Knowledge Sharing Effort (*KSE*) está intentando desarrollar técnicas y una metodología para construir bases de conocimiento en gran escala que facilite la comunicación entre agentes. El principal resultado es el *KQML* (Knowledge Query and Manipulation Language) un formateador de mensajes y protocolo para el manejo de mensajes que soporta el intercambio de conocimientos entre agentes al tiempo de ejecución. *KQML* puede ser usado como un lenguaje para un programa de aplicación para interactuar con un sistema inteligente o para que dos o más sistemas inteligentes compartan el conocimiento para resolver problemas en forma cooperativa.

3. **Facilidad de transporte**. Mientras que la facilidad de comunicación tiene que ver con lo que el agente está comunicando, la facilidad de transporte tiene que ver con la forma en que el agente se está comunicando. La facilidad de transporte permite el movimiento de un agente desde un ambiente de ejecución a otro para llevar a cabo una tarea. También permite la distribución de agentes no inicializados (los agentes inicializados son los que comienzan a ejecutar su tarea dentro de la máquina cliente, y requieren continuarla en la máquina servidor, los no inicializados son aquellos que comienzan su tarea en la máquina servidor). La facilidad de transporte generalmente soporta protocolos de transmisión de datos establecidos tales como *HTTP*, *SMTP*, y *TCP/IP*.

4. **Facilidad para empacar**. Esto provee un método estándar para empacar agentes junto con su información asociada. Sin tomar en cuenta la estructura interna, todos los agentes deben encapsular el estado, la autenticidad, y los

objetivos de la información así como sus capacidades, y la metodología usada en la planeación del manejo de la información. A la fecha ninguna facilidad estándar para empacar ha emergido.

5. **Seguridad integrada**. La identificación y "huellas digitales" de los agentes es un atributo intrínseco de la infraestructura de la arquitectura de un agente. La seguridad es un tema de infraestructura amplio el cual aplica a todos los componentes previamente discutidos.

La infraestructura del agente debe proveer un método seguro inherente para determinar el propietario y el lugar de origen del agente. En la mayoría de las aplicaciones, cada agente posee un "pasaporte" el cual cifrara esta información con el objetivo de que no sea alterada y pueda ocasionar posibles daños.

Los métodos de seguridad utilizados en la infraestructura de un agente deben considerar también la protección de recursos (no abusar de las computadoras o redes), privacidad de la información, y procedimientos y reglas de terminación del agente. Generalmente, cada aplicación de agentes, utiliza sus propios protocolos de seguridad para manejar los requerimientos específicos de la aplicación.

Investigaciones Actuales

Actualmente se hace investigación sobre software de agentes en al ámbito universitario. El Laboratorio de Medios del MIT se está esforzando por aplicar los preceptos de la teoría de la Inteligencia Artificial en agentes destinados a trabajar en el World Wide Web. El Grupo de Agentes Autónomos ha llegado a ser uno de los líderes en el desarrollo de agentes para el World Wide Web.

El grupo de la de la Universidad de Washington ha desarrollado un "*softbot* para Internet" que puede aprender y adaptarse mientras cumple con tareas para su usuario. El software puede llevar solicitudes de su usuario en un lenguaje de alto nivel y utilizar el shell de UNIX para ejecutar sus rutinas.

Otras instituciones como Carnegie Mellon, la Universidad de Maryland en Baltimore, la Universidad de Michigan, y Stanford han establecido actividades de investigación, directamente con varias universidades.

Los dominios de aplicación en la cual las soluciones que brindan los agentes están siendo aplicadas o investigadas son administración de flujos de trabajo, administración de sistemas y de redes, control de tráfico aéreo, reingeniería de procesos de negocios, minería de datos, recuperación y administración de información, comercio electrónico, educación, asistentes digitales personales (*PDAs*), correo electrónico, bibliotecas digitales, control, bases de datos inteligentes, etc.

Conclusión

Consideramos que la administración de sistemas y de redes es una de las principales áreas de aplicación que pueden ser mejoradas utilizando la teoría de agentes. Las arquitecturas de agentes han existido en esta área por algún tiempo, pero éstos son generalmente de "funciones fijas" más que de agentes. Sin embargo, pueden ser usados para mejorar el software para la administración de sistemas. Por ejemplo, pueden ayudar a filtrar y a llevar a cabo acciones automáticas en un alto nivel de abstracción o bien pueden ser usados para detectar y reaccionar a patrones en el comportamiento de un sistema. Más aún, pueden ser usados para manejar grandes configuraciones dinámicamente.

A continuación enumeramos una serie de características que tienen los agentes las cuales consideramos atractivas para la implementación de sistemas de administración para redes y sistemas:

- **Tolerante a fallas**. Un agente es tolerante a fallas debido a que el control se encuentra distribuido en cada uno de los agentes ejecutandose en cada una de las máquinas. Es decir, cada agente tiene una misma prioridad y responsabilidad dentro del sistema. Si algún agente deja de funcionar no compromete el funcionamiento de los demás agentes y por lo tanto de todo el sistema. El agente debe sobrevivir a una caída del sistema sin tener que reconstruir su base de conocimiento al reiniciar el sistema. Este es un aspecto de suma importancia ya que la disponibilidad de los sistemas muchas veces se ve afectada por eventos que no se tienen previstos.

- **Sencillo en su implementación.** Cada agente debe realizar alguna tarea sencilla, de tal forma que cada uno de los agentes en sí mismo debe ser simple en su representación interna.

- **Producen poco overhead**. Como los agentes deben ser sencillos y pequeños no deben implicar una gran carga para la máquina que los está ejecutando (tanto en consumo de procesador como en memoria).

- **Flexible**. Como cada agente realiza una tarea especifica la cual debe ser lo más pequeña posible, es relativamente sencillo sustituir algún agente para que el sistema realice alguna otra tarea diferente.

- **Escalable**. Sólo es necesario incluir otros agentes cuando se realicen otras funciones diferentes para que el sistema sea capaz de realizar nuevas tareas.

La popularidad del uso de agentes llegara a conformar con seguridad una moda en la cual un usuario de software será llevado a decidir entre un sistema que tiene agentes y otro que no los tiene, (como se ha visto en la actualidad con los paquetes que tienen ayudas gráficas o son orientados a objetos, en donde el usuario se inclina más por los paquetes que incluyen estas tecnologías o facilidades que por aquéllos que se muestran obsoletos al carecer de ellas.

El siguiente capítulo muestra una visión general de los diferentes Frameworks para software de agentes desarrollados en Java que consideramos nos pueden ser útiles para el desarrollo del sistema.

Capítulo 4

Frameworks para Agentes

En esta sección se presentan algunos de los frameworks para agentes más conocidos y que presentan puntos de vista distintos con respecto a implementar agentes de software y a la forma de ser utilizados.

Introducción

Actualmente existen varias arquitecturas de agentes implementadas en diversos lenguajes tales como *Telescript*, *TCL/TK* y Java.

Cabe aclarar que estos lenguajes son relativamente nuevos y han mostrado su facilidad de uso sobre Internet en diversas plataformas. *Telescript* se ha distinguido por ser un lenguaje dirigido explícitamente para la creación, manejo, traslado y monitoreo de agentes, no obstante este lenguaje no es de dominio público y por lo tanto su uso ha sido restringido a proyectos especiales de compañías tales como AT&T y GeneralMagic, dado que tiene un costo elevado. *TCL/TK* es un lenguaje sencillo y de fácil uso para el desarrollo de prototipos y aplicaciones académicas, no obstante el principal problema con el lenguaje es su baja velocidad para ejecutar las aplicaciones finales de los usuarios. Por lo anterior, en este trabajo se da preferencia a la utilización de frameworks basados en Java, por ser ampliamente disponible sobre Internet, por su facilidad de portabilidad y por ser gratuito.

Antes de describir los frameworks cabe mencionar algunas de las 2 extensiones que actualmente tiene Java para cómputo distribuido: Object Serialization y Remote Method Invocation.

Object Serialization

Object Serialization permite la codificación de objetos y la disponibilidad de estos a través de un flujo de bytes, además de soportar la reconstrucción complementaria de los mismos. Object Serialization es usado para proporcionar un cierto grado de persistencia y para la comunicación por medio de sockets o *RMI* (Remote Method Invocation). En Java esto es proporcionado por medio de un conjunto de clases que extienden sus clases centrales de entrada y salida para que soporten objetos. La codificación por omisión de los objetos protege datos privados y transitorios. Además, una clase puede implementar su propia codificación externa.

RMI (Remote Method Invocation)

Los sistemas distribuidos requieren que el procesamiento se lleve a cabo en diferentes espacios de direccionamiento, inclusive en diferentes máquinas.

Además, es necesario que los procesos sean capaces de comunicarse. Si lo que se necesita es un mecanismo de comunicación básica, Java soporta *sockets*, los cuales son flexibles y suficientes para comunicaciones en general. Sin embargo, los *sockets* requieren que el cliente y el servidor trabajen con protocolos de nivel de aplicación para codificar y decodificar mensajes necesarios para realizar el intercambio de información, el diseño de tales protocolos es complicado y difícil de utilizar.

Una alternativa a los *sockets* son las RPCs (Remote Procedure Calls) las cuales hacen una abstracción de la interfaz de comunicación al nivel de llamadas a procedimientos. En lugar de trabajar directamente con *sockets*, el programador tiene la idea de estar haciendo llamadas a procedimientos locales, cuando, de hecho, los argumentos de la llamada son empaquetados y mandados al destino remoto de la llamada. Los sistemas *RPC* codifican sus argumentos y valores de retorno usando una representación de datos externa, tal como *XDR* (External Data Representation). *RPC*, sin embargo, no funciona adecuadamente en sistemas de objetos distribuidos, donde la comunicación entre objetos a nivel de programa residiendo en diferentes espacios de direccionamiento es necesario. Con el objeto de mantener la semántica de la invocación de los métodos los sistemas de objetos distribuidos requieren la utilización de *RMI* (remote method invocation). En tales sistemas, un objeto substituto local maneja la invocación de un objeto remoto.

RMI permite crear aplicaciones distribuidas, en las cuales los métodos de objetos remotos pueden ser invocados desde otras máquinas virtuales Java. Un programa Java puede hacer una llamada a un objeto remoto una vez que éste obtiene una referencia al objeto al buscar el objeto remoto en el servicio de nombres *bootstrap* proporcionado por el *RMI*, o al recibir la referencia como un argumento. Por otra parte un cliente puede llamar a un objeto remoto en un servidor, y este servidor puede ser un cliente de otros objetos remotos. *RMI* utiliza Object Serialization para parámetros *marshal* y *unmarshal* y no trunca tipos, soportando verdadero polimorfismo.

Este *RMI* ha sido diseñado para operar en el ambiente Java. Se puede hacer uso de otros sistemas *RMI* para manejar objetos Java, no obstante, estos sistemas carecen de una buena integración con el ambiente Java debido a su necesidad de interoperar con otros lenguajes. Por ejemplo, *CORBA* parte de un ambiente heterogéneo, un ambiente multi-lenguaje, por lo que debe contar con un modelo de objetos independiente del lenguaje. En contraste, el sistema *RMI* de Java asume un ambiente homogéneo como lo es la máquina virtual, por lo que el sistema puede tomar ventaja del modelo de objetos de Java cuando es posible.

Características del sistema:

- Soporte para invocación remota de objetos en diferentes máquinas virtuales.
- Soporte de *callbacks* desde servidores a *applets*.
- Integra el modelo de objetos distribuidos dentro del lenguaje Java en una forma natural mientras mantiene la semántica de los objetos.

- Preserva el mecanismo de seguridad del ambiente de ejecución de Java.
- Garbage collection distribuido para la eliminación de objetos remotos.
- Activación de objetos persistentes para atender peticiones.

Descripción de los Frameworks

A continuación dos de las principales implementaciones de agentes utilizando Java son mostradas. La primera de ellas con un enfoque más hacia la parte de inteligencia artificial y la segunda hacia la creación de agentes móviles.

JAT (Java Agent Template)

El JAT provee un completo *template* funcional escrito en su totalidad en Java para la construcción de software de agentes, los cuales se comunican uno a uno con una comunidad de otros agentes distribuidos a través de Internet. Aunque parte del código que define a un agente es portable, los agentes creados con JAT no son migratorios, más bien estos tienen una existencia estática sobre una máquina. Este comportamiento está en contraste a otras tecnologías de agentes. (Sin embargo, usando el *RMI*, los agentes JAT podrían dinámicamente migrar a otra máquina). Actualmente, todos los mensajes de agentes usan *KQML* como un protocolo de alto nivel o encapsulador de mensajes. El JAT incluye funcionalidad para el intercambio dinámico de "recursos", lo cual puede incluir clases de Java (ejem. nuevos lenguajes e intérpretes, servicios remotos, etc.), archivos de datos e información en línea dentro de los mensajes KQML..

JAT puede ser ejecutado como una aplicación *stand-alone* o como un *applet* a través del *appletviewer* (en visualizadores como Netscape no trabaja adecuadamente debido a las restricciones en el uso de red y lectura y escritura de archivos). Ambas configuraciones soportan agentes con o sin interfaz gráfica. La coordinación entre estos es llevada a cabo por medio de un servidor de nombres de agentes. La arquitectura del JAT fue especialmente diseñada para permitir el remplazo y especialización de componentes que agreguen funcionalidad, tales como GUIs, mensajes de bajo nivel, interpretación de mensajes y manejo de recursos. Consecuentemente, el JAT puede ser usado como una plataforma para la construcción de un amplio tipo de agentes para diferentes aplicaciones.

EL JAT 0.3 incluye un conjunto de paquetes centrales (***JavaAgent.context***, ***JavaAgent.agent*** y ***JavaAgent.resource***), paquetes que soportan un paradigma de servicios remotos (***RemoteService.context***, ***RemoteService.agent***, ***RemoteService.resource*** e ***ImageSelector***) y los paquetes que implementan ejemplos incluidos en la distribución del JAT (***test***, ***FDATI*** and ***SModeling***). La documentación proporcionada con el framework es extensa.

Los siguientes directorios están disponibles cuando se instala el sistema:

clases: Contiene todos los archivos .java y .class así como algunos Makefiles.
scripts: *Scripts* en cshell para simplificar la ejecución.
working. Contiene directorios en los cuales los agentes pueden almacenar archivos durante su ejecución, incluyendo archivos de log.
files: Contiene directorios para los archivos y clases que serán compartidas entre los agentes
api: documentación generada por el javadoc.
applet: Archivos *html* para la ejecución de *applets*.
documentation: Documentación en *html*.

Todos los agentes JAT mantienen tanto clases como datos usando objetos, los cuales con subclases de la clase ***JavaAgent.resource.Resource***.

Los recursos de clase incluyen lenguajes (manejadores de protocolo esencialmente, habilita la opción de poder analizar un mensaje y provee algunas semánticas de alto nivel) e intérpretes (esencialmente manejadores de contenido, proporciona una especificación procedural de cómo un mensaje, construido de acuerdo a un *ontology* específico, debería ser interpretado). (El intérprete es especificado usando el campo "*ontology*" del mensaje *KQML*). Se asume que la implementación de cada clase ***Interpreter*** está basada en alguna especificación formal (*ontology*) de las semánticas de mensaje. Una propiedad del JAT da la posibilidad de que estos recursos puedan ser dinámicamente intercambiados entre agentes de una forma "*just-in-time*" (una subclase de ***Resource***, ***RetrievalResource*** es utilizada para este propósito). Esto permite a un agente procesar correctamente los mensajes, cuyo lenguaje e intérprete son desconocidos, al adquirir las clases e intérprete de forma dinámica.

Los agentes que se encuentran en los paquetes base (***Agent*** y *ANS*) no están pensados para trabajar con ellos tal y como están, estos deben ser subclasificados para proporcionar alguna funcionalidad específica. El JAT incluye un ejemplo de especialización, el cual soporta el uso de servicios remotos, manejados por ***ServicesAgents***, y por un grupo distribuido de ***ClientAgents***. La implementación de esta arquitectura de servicio remoto está contenida en los paquetes ***RemoteService***. Un agente ***ServiceBroker*** (*ANS*) sirve como un repositorio para servicios registrados, además de su funcionalidad normal de *ANS*. Cada ***ServiceAgent***, cuando inicia, registra el grupo de servicios que éste administra con el ***ServiceBroker***. Los ***ClientAgents***, cuando inician, automáticamente obtienen la lista de servicios disponibles desde el ***ServiceBroker***. Cada servicio disponible puede ser instalado y ejecutado por un ***ClientAgent*** por medio de la interacción con el ***ServiceAgent***. Esta arquitectura es genérica y puede soportar una amplia variedad de aplicaciones las cuales pueden ser representadas por el paradigma de servicios remotos.

A continuación se muestra la arquitectura del template:

Community-level architecture
ANS

Agents
Agent-level architecture
Context
GUI
Communication Interface
Agent
Message Handler
Resource Manager

Aglets Workbench

Aglets Workbench es un ambiente para la construcción de aplicaciones basadas en red que usan agentes móviles para la búsqueda, acceso y manipulación de datos. Según la arquitectura de este framework los agentes pueden ser enviados desde cualquier computadora y transportados a otra para su ejecución. Al llegar a la máquina huésped, los agentes presentan sus credenciales y obtienen acceso a los servicios y datos locales. La computadora remota puede también servir como un intermediario al agrupar agentes con intereses similares y metas compatibles, brindando de esta forma un lugar de reunión en el cual los agentes puedan interactuar. Además de lo anterior, el framework proporciona lo siguiente:

- **Acceso a bases de datos.**
- **Búsquedas.**
- **Una forma estándarizada y segura para la comunicación.**
- **Movilidad.** Cuando un agente se mueve, el objeto completo es enviado; esto es tanto su código, sus datos y su estado de ejecución e itinerario de viajes son trasladados conjuntamente.
- **Autonomía.** El agente móvil contiene suficiente información para decidir que hacer, a donde ir, y cuando ir.
- **Asincronía.** El agente tiene su propio thread de ejecución y puede ejecutarse de forma asincrona.
- **Interacción local.** El agente interactúa con otros agentes móviles o estacionarios localmente a través de mensajes y el uso de una arquitectura de pizarrón. Si es necesario éste puede mandar agentes mensajeros o contratados, para facilitar la interacción remota.
- **Operación desconectada.** El agente puede ejecutar sus tareas ya sea que la conexión de la red esté abierta o cerrada. Si la conexión de la red está cerrada y éste necesita trasladarse puede y debe esperar hasta que la conexión sea reabierta.

- **Ejecución en paralelo.** Más de un agente puede ser enviado a diferentes sitios para ejecutar tareas en paralelo.

Este framework cuenta con una clase *Aglet*, la cual proporciona una forma muy conveniente para crear agentes definidos por el usuario que heredan propiedades y funciones por omisión para los agentes móviles. Estas funciones incluyen:

- Un esquema de nombres únicos globalmente para agentes.
- Un itinerario de viaje para especificar patrones de viaje complejos con múltiples destinos y manejo automático de fallas.
- Un mecanismo de pizarrón permite a múltiples agentes colaborar y compartir información en forma asincrona.
- Un esquema de envío de mensajes que soporta comunicación punto a punto tanto síncrona como asincrona entre agentes.
- Un cargador de clases de agentes de red que permite que el código de bytes y la información del estado del agente viaje a través de la red.
- Un contexto de ejecución que provee a los agentes con un ambiente uniforme independiente del sistema de computación actual sobre el que ellos se están ejecutando.

Otra forma en la cual el framework soporta el desarrollo de agentes es a través de la introducción del uso de patrones. El diseño y desarrollo basado en patrones ha mostrado ser muy útil en un gran número de áreas. Este framework tiene un número de patrones de uso de agentes de alto nivel. Estos patrones de uso describen relaciones comunes entre agentes tales como ***Master-Slave*** (maestro-esclavo), ***Messenger-Receiver*** (mensajero-receptor), y ***Notifier-Notification*** (notificador-notificado). Estos patrones son presentados como un conjunto de clases que pueden ser usados como *templates* (esqueletos) por el desarrollador de agentes.

El *API* (Application Program Interface) del framework está completamente documentado y permite fácilmente la creación de agentes móviles para sistemas de información. Algunas de las principales características que tiene el framework son:

Acceso a Datos corporativos

El acceso a bases de datos corporativos es esencial en muchas de las aplicaciones de agentes móviles. Aglets Workbench tiene varios paquetes para acceso a datos, incluyendo JDBC/DB2 y JoDax. Estos paquetes evitan que el desarrollador de agentes tenga que hacer uso de métodos primitivos para accesar a bases de datos sin importar si son locales o remotas Actualmente JoDax trabaja únicamente en Windows95.

ATP (Agent Transfer Protocol)

El *ATP* (protocolo para transferencia de agentes) es usado para transferir agentes sobre la red. *ATP* es un protocolo a nivel de aplicación para sistemas de información distribuidos basados en agentes. Este protocolo está basado sobre Internet y el uso de URLs para la localización de recursos de agentes. *ATP* ofrece un protocolo uniforme y transparente, independiente de la plataforma que se esté utilizando para la transferencia de agentes entre computadoras.

Mientras que los agentes móviles pueden ser escritos en muchos diferentes lenguajes y por una variedad de sistemas de agentes específicos del proveedor, *ATP* ofrece la oportunidad para manejar la movilidad en una forma uniforme y general.

Por ejemplo, cualquier máquina huésped de agentes tiene un nombre único y simple, independiente del conjunto de sistemas de agentes específico del vendedor de la máquina. *ATP* provee un mecanismo de transporte de agentes uniforme y da facilidades al usuario para solicitar agentes de forma estándar a través de toda la red.

La primera versión de *ATP* que es la que se utiliza actualmente, está enteramente escrita en Java e implementada como un paquete completamente independiente y documentado dentro del Aglets Framework. El paquete *ATP* está formado por un conjunto de clases altamente portables que proveen un *API* para crear daemons, conectarse a sitios *ATP*, y generar solicitudes y respuestas *ATP*. Este paquete es independiente de cualquier implementación de agentes en particular.

Administrador de Agente Visual

Tahiti es un manejador de aglets visual basado en el Aglets Framework. *Tahiti* usa una interfaz de usuario gráfica para monitorear y controlar la ejecución de los aglets en la computadora local. A través de una interfaz *drag and drop* uno puede hacer que dos *aglets* se comuniquen entre si, o mandar a un *aglet* a un sitio en particular. *Tahiti* además de ser una herramienta de administración del sistema; es una herramienta de escritorio para que los usuarios de los *aglets* manipulen a sus agentes de la misma forma en que usan un *visualizador* de *WWW* como herramienta fundamental.

Aglets sobre el WWW

El *WWW* es una importante infraestructura para los agentes móviles. Los agentes desarrollados con Aglets Workbench podrán ser embebidos en páginas de *WWW* de tal forma que permiten a los usuarios mandar a estos agentes a Internet directamente desde el visualizador del WWW.

Aglets Workbench incluirá un lanzador de agentes de *WWW* llamado *Fiji*. *Fiji* es un *applet* capaz de crear un *aglet* o traer un *aglet* existente dentro del visualizador de *WWW* del usuario. El *applet Fiji* simplemente toma un *URL* de agente como su parámetro y puede fácilmente ser embebido en una página de *HTML*, como cualquier *applet*. Exactamente como otro *applet* todo el software requerido será dinámicamente cargado al visualizador. El usuario no tiene explícitamente que bajar e instalar Aglets Workbench para poder usar los agentes.

Si el servidor de *WWW* es acompañado con un daemon *ATP* que soporta *aglets*, los *applets Fiji* son capaces de mandar *aglets* a este servidor de *WWW* para realizar búsquedas remotas o monitoreos de sitios *WWW* sin tener una conexión permanente.

Seguridad

La seguridad es un punto importante para los usuarios de agentes móviles. Por un lado los agentes pueden ser usados para un gran cantidad de actividades en beneficio del usuario, no obstante, por otra parte ellos podrían llegar a ser una tierra fértil para la creación de virus. Recibir agentes desconocidos de la red es potencialmente una invitación abierta para una gran cantidad de problemas.

El Aglets Framework soporta un modelo de seguridad por capas. La primera capa es el sistema del lenguaje mismo. Los fragmentos de código importados en los agentes son sujetos a una serie de pruebas para asegurar que el formato del código que transportan es correcto, y finaliza con una serie de revisiones realizadas por el verificador de *bytecode* de Java.

En la siguiente capa está el administrador de seguridad, el cual permite a los usuarios del Aglets Framework implementar sus propios mecanismos de seguridad. *Tahiti* implementa un administrador configurable que proporciona un alto grado de seguridad para la computadora huésped y su propietario. La configuración por omisión de seguridad es muy restrictiva. Cualquier intento por parte de un agente para accesar un archivo para el cual el acceso no ha sido proporcionado será considerado como una violación de seguridad, y al agente no se le permitirá el acceso.

En la tercera y ultima capa está el *API* de seguridad de Java, el cual es un framework sencillo para el desarrollador de agentes. Éste incluye cierta funcionalidad de seguridad en sus agentes. Dicha funcionalidad incluye a su vez el uso de técnicas de *criptografía*, firmas digitales, cifrado, y autentificación.

Estandarización

Tanto el *ATP* como el Aglets Workbench han sido propuestos a la *OMG* (Object Managment Group) Grupo de Administracion de Objetos, para su aceptación como el estándar para agentes móviles.

Aún más, la especificación del *ATP* es de dominio público. El protocolo ha sido modelado sobre *HTTP*, y propone una forma estándar de transportar agentes independientemente de cualquier implementación particular. Actualmente Aglets Workbench puede ser obtenido gratuitamente desde Japón.

Algunas de las extensiones que tendrá el framework y que actualmente están en desarrollo son:

- *Un ambiente de desarrollo visual*. Los constructores visuales actualmente son esenciales para mostrar la productividad del programador, teniendo que un 40% del desarrollo de un sistema es dedicado a la presentación. Mediante el uso de una metáfora *drag-and-drop*, el constructor visual en Aglets Workbench permitirá visualmente combinar componentes gráficos y no gráficos dentro de agentes móviles personalizados. El constructor visual llamado *Tazza* hará fácil el desarrollo de aplicaciones de agentes con comportamiento móvil e interfaces gráficas.

En resumen el Aglets Workbench está conformado por los siguientes componentes para construir agentes y aplicaciones:

ATP (Agente Transfer Protocol) El *ATP* es el protocolo estándar para transferir agentes móviles. Éste usa *URL* para la localización de recursos. Este framework consiste de un conjunto de clases las cuales permiten la creación de daemons *ATP*, abrir conexiones a recursos *ATP*, procesamiento de solicitudes *ATP*, y generación de respuestas *ATP*.

AF (Aglets Framework) Un aglet es un agente móvil. Además de la clase *Aglet* este framework consiste de clases las cuales proporcionan el contexto de ejecución, identificación, *proxy*, e itinerario para un *aglet*. Éste también tiene un cargador responsable de la carga y descarga de Aglets cuando estos se desplazan.

Patrones de *Aglets*. Estos consisten de los siguientes patrones:

Mensajero-receptor. Éste permite crear un *aglet* mensajero y enviarlo a un aglet receptor para repartir mensajes.

Maestro-esclavo. Éste permite a un *aglet* maestro crear un *aglet* esclavo y enviarlo a un sitio remoto para ejecutar alguna tarea. Una vez que el *aglet* esclavo completa su itinerario, regresa y reporta los resultados al *aglet* maestro.

Notificador-Notificación Permite crear un *aglet* notificador y enviar éste a un sitio remoto para monitorear eventos de interés. Cada vez que tales eventos ocurran, el *aglet* notificador enviará una notificación a través de un *aglet* mensajero.

Servidor de Aglet. Éste provee una interfaz gráfica de usuario que permite fácilmente al usuario enviar, recobrar y clasificar un *aglet*. Éste también ejecuta funciones de administración tales como administración de seguridad, administración de *threads*, y entrada de agentes al sistema.

Lanzador de *Aglets*. Éste permite al usuario mandar y recobrar un *aglet* desde un *applet*. Éste es un componente clave para permitir a los *aglets* servir como el mecanismo de integración entre *WWW* y *CORBA*.

Componentes para acceso a datos (*JoDax* y *JDBC*)

Conclusión

El AgentTemplate toolkit es una excelente herramienta para la construcción de agentes que requieren el uso de inteligencia artificial. Bajo el AgentTemplate, las interacciones entre agentes están basadas sobre Knowledge Query and Manipulation Language (*KQML*), un lenguaje diseñado específicamente para aplicaciones de inteligencia artificial. Mientras ésta es una solución sofisticada para algunas de las aplicaciones basadas en agentes, los agentes por si mismos son estacionarios y no son apropiados para verdaderas aplicaciones interactivas. Además, el uso de un lenguaje por separado para el intercambio de información es una complejidad que muchas aplicaciones no requieren.

Por otra parte el Aglets Workbech proporciona algunas caracteristicas que consideramos fundamentales para nuestro sistema:

- Objetos móviles y estacionarios.
- Un método para el manejo de objetos, mensajes y datos
- Objetos autónomos y pasivos
- Procesamiento síncrono y asíncrono
- Operaciones con y sin conexión
- Ejecución secuencial y en paralelo

Esta capacidad de movilidad resulta muy conveniente para su uso en el *WWW* y en nuestro sistema de administración, el cual es presentado en el siguiente capítulo. En dicho capítulo se plantea el diseño del sistema y la forma en que se implementó.

Capítulo 5
Desarrollo del sistema

En este capítulo se describe el análisis, diseño e implementación del sistema de administración, tema central de esta tesis. El desarrollo de este capitulo se encuentra dividido en las siguientes secciones:

- Especificación de requerimientos
- Creación de un bosquejo de ¿cómo se desea ver la aplicación?
- Análisis de las clases del *framework*
- Identificación de las clases del sistema
- Definición de responsabilidades
- Colaboración entre objetos
- Creación del prototipo
- Extensión del prototipo
- Pruebas de funcionamiento

Análisis y Diseño

Especificación de requerimientos

En esta etapa se plantea el problema que se desea resolver y los objetivos y metas a alcanzar.

En el primer capítulo mencionamos la importancia de contar con un sistema de administración distribuido, multiplataforma, flexible, seguro y con un GUI sencillo y fácil de manejar para el control de los equipos de cómputo en una empresa. Estas son las principales características que se deben tomar en cuenta para la creación del sistema.

Dado que las labores de administración en sistemas de cómputo son tantas y tan diversas, y de acuerdo con nuestra experiencia como administradores de sistemas tomamos la decisión de que el sistema debería contar con facilidades que nos permitieran ejemplificar de una manera clara el uso de distintos tipos de aglets en la solución de problemas comunes, tales como: aglets para monitorear archivos de configuración, para actualizar archivos, y para recopilar información importante de cada una de las máquinas (inventario). El sistema deberá permitir el monitoreo tanto de estaciones de trabajo como de PCs, así como la creación de reportes en un formato fácil de manipular y desplegar, como lo es el formato de las páginas *HTML*. Éste ultimo brindará la facilidad al usuario de ver los documentos con cualquier *visualizador* como Netscape o Internet Explorer.

La decisión del equipo en el cual funcionará el sistema se tomó con base en el equipo disponible en el laboratorio de la Maestría en Ciencias de la

Computación de la Universidad Nacional Autónoma de México, no obstante, el sistema está pensado para funcionar en cualquier otra plataforma donde Java esté disponible.

En el caso del monitoreo de archivos el sistema deberá ser capaz de monitorear en forma paralela tanto los archivos como las máquinas que así se le especifiquen.

En los casos de inventario y actualización el tiempo no es una prioridad por lo que estas tareas se pueden realizar en forma secuencial.

El sistema deberá ser capaz de responder ante eventualidades que no son predecibles como problemas en la red y/o caídas de algún cliente o del servidor mismo.

Deberá contemplar aspectos de seguridad para la integridad de la información almacenada en las máquinas administradas.

El sistema no deberá representar una carga muy grande para las máquinas, además de que no interferirá con otros sistemas ejecutándose tanto en los clientes como en el servidor.

Deberá ser lo suficientemente modular para permitir la integración de una mayor funcionalidad de manera simple y congruente con la forma de operar del sistema.

Creación de un bosquejo

Para poder tener una mejor idea de la forma en que el sistema funcionará el diseñador deberá crear un bosquejo de la interfaz gráfica del sistema y de la funcionalidad que deberá tener ésta.

La aplicación contará con una ventana principal a través de la cual el usuario podrá administrar los diferentes tipos de agentes, y desde la cual podrá invocar los diferentes diálogos necesarios para la configuración de los aglets.

Es importante aclarar que los dibujos que se muestran a continuación sufrieron diversas modificaciones antes de llegar a su representación gráfica final. Esta fase ayudará al diseñador a encontrar alguna funcionalidad oculta no prevista en la parte de requerimientos o bien para replantear algunas de ellas, además de poder fijar con claridad a lo que se desea llegar. Aquí no se consideran las limitaciones que en un momento dado podríamos encontrar debido al lenguaje utilizado para la implementación.

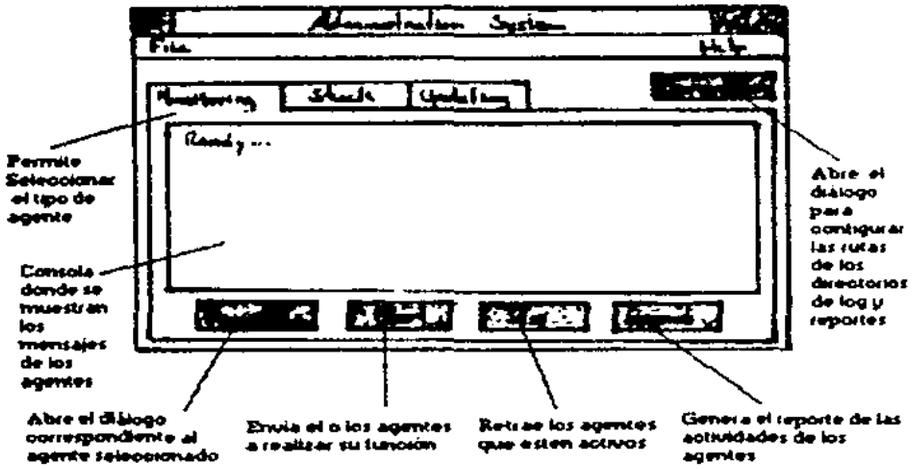


Figura 10 Ventana principal de la aplicación

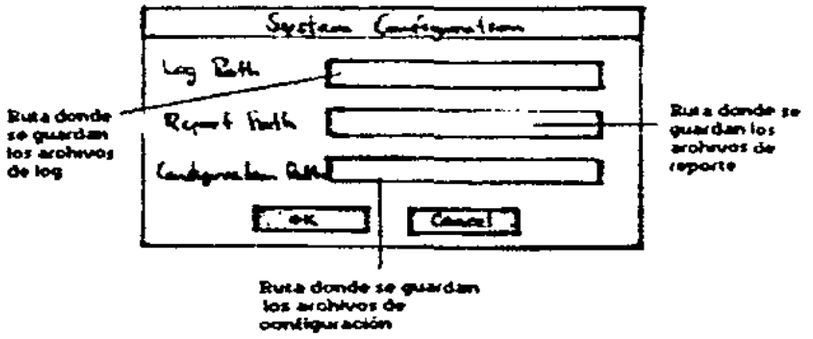


Figura 11 Dialogo de configuración de la aplicación

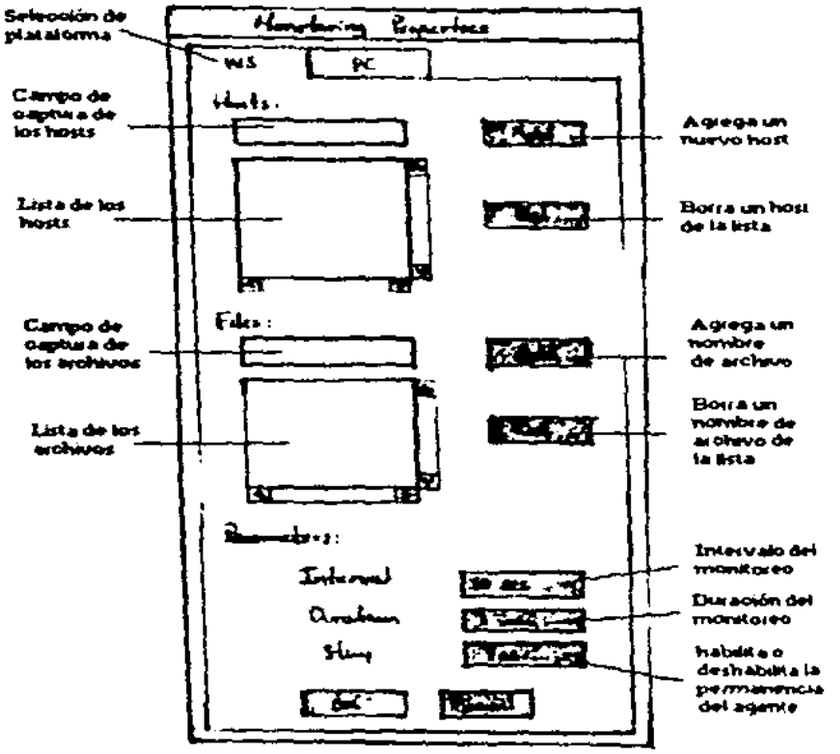


Figura 12 Diálogo de configuración de monitoreo

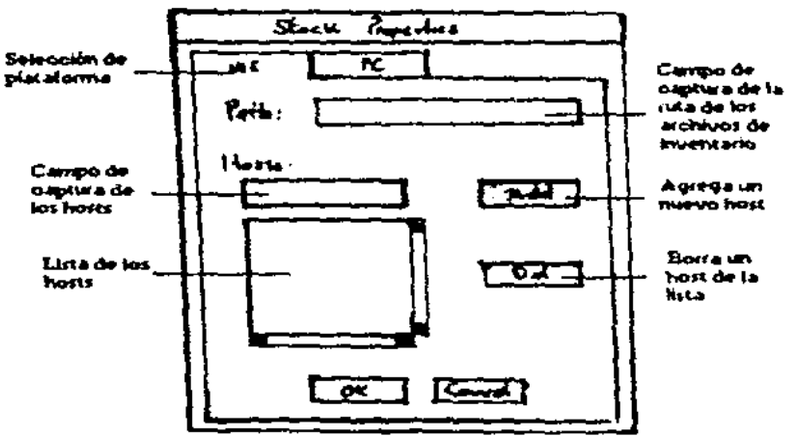


Figura 13 Diálogo de configuración de inventario

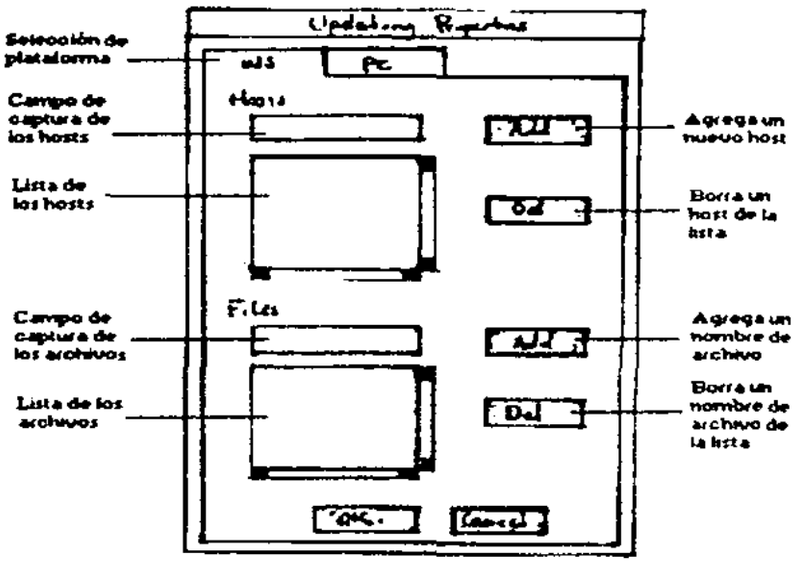


Figura 14 Diálogo de configuración de actualización

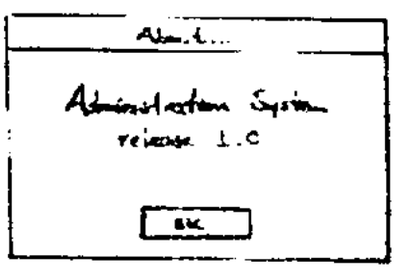


Figura 15 Diálogo de información

Análisis de las clases del framework (marco de trabajo)

Para poder definir claramente las clases que necesitamos es necesario estudiar las ya existentes dentro del marco de trabajo de los *Aglets*.

Elementos del API Java-Aglet

J-AAPI es un estándar propuesto para manejar *aglets* y su ambiente. Contiene métodos para inicializar un *aglet*, manejar mensajes, enviar, retraer, desactivar/activar, duplicar, y desechar *aglets*. J-AAPI es simple, flexible, y estable.

Las interfaces y clases del *J-AAPI* son las siguientes:

Interfaces

- ***AgletContext***
- ***Future***

Clases

- ***Aglet***
- ***AgletIdentifier***
- ***AgletProxy***
- ***Arguments***
- ***Itinerary***
- ***Message***

El Modelo de Objetos de los Aglets

Primero describiremos el modelo de objetos sobre el cual se fundamenta el *J-AAPI*. El modelo de objetos del aglet define un conjunto de abstracciones y el comportamiento necesario para cumplir el propósito de la tecnología de agentes tanto en Internet como en Redes de Área Amplia. Las abstracciones clave encontradas en este modelo son *aglet*, contexto, *proxy*, mensaje, itinerario, e identificador:

- Un *aglet* es un objeto de Java móvil que visita las máquinas habilitadas para aglets en una red de computadoras. Es autónomo, debido a que se ejecuta sobre su propio thread de ejecución después de llegar a una máquina, y reactivo, gracias a su capacidad de responder a los mensajes que recibe.

- Un contexto es el lugar de trabajo del *aglet*. Es un objeto estacionario que proporciona un medio para mantener y manejar aglets en un ambiente de ejecución donde el sistema huésped está asegurado contra aglets maliciosos. Un nodo en una red de computadoras puede hospedar múltiples contextos.

- Un *proxy* es un representante de un aglet. Sirve como un escudo que protege al aglet de accesos directos a sus métodos públicos. También da transparencia a la localización del aglet, es decir, puede ocultar la localización real del aglet.

- Un mensaje es un objeto intercambiado entre aglets. Permite la transmisión de mensajes síncronos y asíncronos entre aglets. La transmisión de mensajes

puede ser usada por los aglets para colaborar e intercambiar información libremente.

- Un itinerario es un plan de viaje del *aglet*. Proporciona una abstracción conveniente para patrones complicados de viaje y de encaminamiento.

- Un identificador está ligado a cada *aglet*. Este identificador es globalmente único e inmutable a través del tiempo de vida del *aglet*.

El comportamiento soportado en el modelo de objetos del *aglet* incluye creación, duplicación, envío, retracción, desactivación, activación, deshacerse de, e intercambio de mensajes:

- La creación de un *aglet* toma lugar en un contexto. Al nuevo *aglet* se le asigna un identificador, es insertado en el contexto, e inicializado. El *aglet* comienza su ejecución tan pronto como éste ha sido exitosamente inicializado.

- La duplicación de un aglet produce una copia casi idéntica del aglet original en el mismo contexto. Las únicas diferencias son los identificadores asignados y que la ejecución reinicia en el nuevo aglet. Los threads de ejecución no son duplicados.

- El envío de un aglet de un contexto a otro lo removerá de su contexto actual y lo insertará dentro de su contexto destino, donde reiniciará la ejecución (los threads de ejecución no migraran). Se dice que el aglet ha sido empujado a su nuevo contexto.

- La retracción de un *aglet* lo jalará (removerá) desde su contexto actual y lo insertará dentro del contexto desde el cual la retracción fue solicitada.

- La desactivación de un *aglet* es la capacidad para removerlo temporalmente de su contexto actual. La activación de un *aglet* lo restaurará en un contexto.

- La transmisión de mensajes entre agentes implica enviar, recibir, y manejar mensajes síncronamente y asíncronamente.

- Un mecanismo automático de nombrado asigna identidades inmutables a los nuevos *aglets*. Estas identidades deberán garantizar ser globalmente únicas.

La Clase *Aglet*.

La clase ***Aglet*** es la clase principal en J-AAPI. Ésta es la clase abstracta que el desarrollador de *aglets* usa como clase base al crear sus propios *aglets*. La clase ***Aglet*** define métodos para controlar su propio ciclo de vida, métodos para duplicación, envío, desactivación, y deshacerse de sí mismo. También define métodos que deben ser sobrescritos en sus subclases por el programador, y proporciona los "ganchos" necesarios para personalizar el comportamiento del aglet. Estos métodos son sistemáticamente invocados por el sistema cuando

ciertos eventos toman lugar en el ciclo de vida de un *aglet*. La siguiente tabla muestra la relación entre estos eventos del ciclo de vida y los métodos en el *aglet*.

El evento	El evento toma lugar	Después de haber tomado lugar
Creación		`onCreation()`
Duplicación	`onCloning()`	`onClone()`
Envío	`onDispatching()`	`onArrival()`
Retracción	`onReverting()`	`onArrival()`
Deshacerse	`onDisposing()`	
Desactivación	`onDeactivating()`	
Activación		`onActivation()`
T. Mensajes	`handleMessage()`	

Cuando un *aglet* ha sido creado o cuando éste llega a un nuevo contexto, se le da su propio thread de ejecución a través de una invocación al sistema de su método `run()`. Se puede ver esta invocación como un medio a través del cual se le da al *aglet* un grado de autonomía. El método `run()` es llamado cada vez que el *aglet* llega o es activado en un nuevo contexto. Se puede decir que el método `run()` llega a ser el punto de entrada principal para el *thread* de ejecución del *aglet*. Al sobreescribir este método se puede personalizar el comportamiento autónomo del *aglet*.

Por ejemplo, se puede utilizar `run()` para permitir que el *aglet* se envíe a si mismo a un contexto remoto. Esto se puede hacer al permitir que el aglet llame a su método `dispatch()` con el Uniform Resource Locator (*URL*) de la máquina remota como el argumento. Este *URL* debe especificar los nombres de la máquina y del dominio del contexto destino, y el protocolo (atp) a ser usado para la transferencia del aglet sobre la red:

```
dispatch(new URL("atp://alguna.máquina.com"));
```

Cuando el método `dispatch()` es invocado el *aglet* desaparece de la máquina en la que se encuentra y reaparecerá en el mismo estado en el destino especificado. Primero, un protocolo de transferencia es utilizado para llevar el *aglet* (*bytecode* y estado de la información) de manera segura sobre la red. Después, la técnica *object serialization* es usada para preservar el estado de la información del *aglet*. Lo que hace es crear una representación secuencial del *aglet* que posteriormente será deserializado para recrear el estado del *aglet*.

Los *aglets* también pueden recibir mensajes. El manejo de mensajes en el *aglet* sigue un esquema *callback*. Cuando un mensaje es enviado al *aglet*, su método `handleMessage()` es invocado con el mensaje actual pasado como argumento. Este método es encargado de manejar los mensajes recibidos. Debe devolver `true` si un mensaje dado es manejado, de otra manera debe devolver `false`. El que envía sabrá de esta manera si el aglet está manejando el mensaje.

La Clase *AgletContext*.

Un *aglet* pasa la mayor parte de su vida en un contexto. Es creado en el contexto, duerme ahí, y muere ahí. Cuando viaja en una red, realmente se mueve de un contexto a otro. En otras palabras, el contexto es un ambiente de ejecución uniforme para *aglets* en un mundo heterogéneo.

La interfaz *AgletContext* es usada por un aglet para obtener información acerca de su ambiente y para enviar mensajes al ambiente, incluyendo otros aglets actualmente activos en ese ambiente. Proporciona medios para mantener y manejar *aglets* ejecutándose en un ambiente donde el sistema huésped está asegurado contra *aglets* maliciosos.

El *aglet* tiene un método para ganar acceso a su contexto actual:

```
context = getAgletContext();
```

Con acceso al contexto, éste puede crear nuevos *aglets*:

```
context.createAglet(...);
```

Y puede retraer (jalar) al contexto actual *aglets* localizados remotamente:

```
context.retractAglet(remoteAgletURL);
```

El *aglet* puede también retraer una lista (enumeración) de *proxies* de los *aglets* presentes en el mismo contexto:

```
proxies = getAgletProxies();
```

El contexto del *aglet* es típicamente creado por un sistema, el cual tiene un *daemon* de red que escucha a la red por *aglets*. Los *aglets* recibidos son insertados dentro del contexto por el *daemon*.

La Clase *AgletProxy*

La clase *AgletProxy* sirve como un escudo para los *aglets*, protegiéndolos del acceso directo a sus métodos públicos.

Un *proxy* es lo que se obtiene del método `createAglet()` en el contexto:

```
AgletProxy proxy = context.createAglet(...);
```

Algunos de los métodos del proxy tales como `clone()`, `dispatch()`, `dispose()`, y `deactivate()`, son usados para controlar el *aglet*. Por ejemplo, para enviar un *aglet* por medio de su *proxy*:

```
proxy.dispatch(new URL("atp://alguna.máquina.com"));
```

Dos métodos, `sendMessage()` y `sendAsyncMessage()`, son usados para enviar mensajes síncronos y asíncronos al *aglet* por medio de su *proxy*. Al enviar un mensaje síncrono, el *thread* del mensaje enviado es suspendido hasta que un resultado es devuelto

```
Object result = proxy.sendMessage(msg);
```

Al enviar mensajes asíncronos, el *thread* del mensaje enviado es devuelto inmediatamente y el resultado es subsecuentemente retraído por medio del objeto future:

```
Future future = proxy.sendAsyncMessage(msg);
...   //No espera por el resultado...
Object result = future.getResult();
```

La Clase *Message*

Los mensajes se distinguen por un tipo de campo de texto nombrado. Este campo es inicializado cuando el mensaje es creado. El segundo parámetro del mensaje constructor es un argumento opcional del mensaje:

```
Message myName = new Message("my name", "Jacob");
Message yourName = new Message("your name?");
```

Una vez que se han creado los objetos mensaje, se pueden enviar al *aglet* por medio de su *proxy*

```
proxy.sendMessage(myName);
String name = (String)proxy.sendMessage(yourName);
```

En el método `handleMessage()` del *aglet* que recibe los mensajes se puede hacer la distinción entre el mensaje `myName()` y el mensaje `yourName()` al probar el tipo de campo de los mensajes que se reciben. De `myName()` se extrae el argumento `name`, y para `yourName()` se coloca el valor de regreso (`setResult()`):

```
public boolean handleMessage(Message msg) {
   if ("my name".equals(msg.kind)) {
      String name = (String)msg.getArg();   // Obtiene el nombre...
      return true;                          // Si manejó el mensaje.
   } else if ("your name?".equals(msg.kind)) {
      msg.setResult("Yina");                // Devuelve su nombre...
      return true;                          // Si manejo el mensaje.
   } else
      return false;                         // No manejo el mensaje.
}
```

La Clase *Itinerary*

El método `dispatch()` de la clase del *aglet* soporta únicamente viajes muy simples de un solo salto para el *aglet*, para *aglets* con planes de viaje más

elaborados, J-AAPI tiene la clase ***Itinerary***. Dar un *aglet* con un itinerario es una forma conveniente de enviarlo a un viaje de múltiples destinos

La clase ***Itinerary*** en *J-AAPI* es una clase abstracta que no se puede instanciar directamente. En vez de eso, se debe crear una clase con itinerario específico con un comportamiento apropiado. ***SeqItinerary*** podría ser una de dichas clases. Ésta ejecuta un plan de viaje directo (secuencial) con un conjunto de visitas consecutivas. Su constructor toma un vector de destinos (*URLs*) como un argumento para inicializar el plan de viaje.

```
Itinerary plan = new SeqItinerary(destVector);
```

Este itinerario es entonces asignado a un *aglet*:

```
someAglet.setItinerary(plan);
```

El *aglet* puede ahora retraer su itinerario actual e invocar el método `go()` del itinerario con la palabra clave `NEXT` para proceder de acuerdo a su plan de viaje:

```
getItinerary().go(Itinerary.NEXT);
```

Las otras palabras clave para el método `go()`, `PREVIOUS`, `FIRST`, y `LAST` se explican por sí mismas. Otras palabras clave pueden ser agregadas para itinearios especializados.

La Clase *AgletIdentifier*

Cada *aglet* tiene asignado una identidad única globalmente que mantiene a través de su tiempo de vida. La clase ***AgletIdentifier*** es una abstracción conveniente para esta identidad. El objeto identificado oculta la representación de la implementación específica de la identidad del *aglet* . El identificador es un objeto inmutable [nota: desde la versión *alpha4*] que se puede retraer directamente desde el *aglet* y su *proxy*.

```
AgletIdentifier aid = proxy.getIdentifier();
```

Teniendo el identificador y un contexto se le puede solicitar al contexto que retraiga el aglet con una identidad específica:

```
proxy = context.getAgletProxy(aid);
```

Identificación de las clases

De acuerdo a las fases anteriores, se procedió a identificar las clases necesarias para la implementación del sistema. Para los nombres de todas las clases relacionadas con el sistema se opto por incorporar siempre como prefijo las siglas "*AAS*" (Sistema de Administración de Aglets), esto con el fin de que queden bien identificadas. Para dividir la complejidad del sistema se opto por crear tres

subsistemas principales: Subsistema para la creación y manipulación de *aglets*, subsistema para la creación de la interfaz gráfica (GUI) y subsistema para la creación de reportes.

Subsistema para la creación y manipulación de aglets

Este subsistema contempla la creación y manipulación de *aglets* sin preocuparse en su representación gráfica. Este subsistema consta de 5 clases que son descritas a continuación:

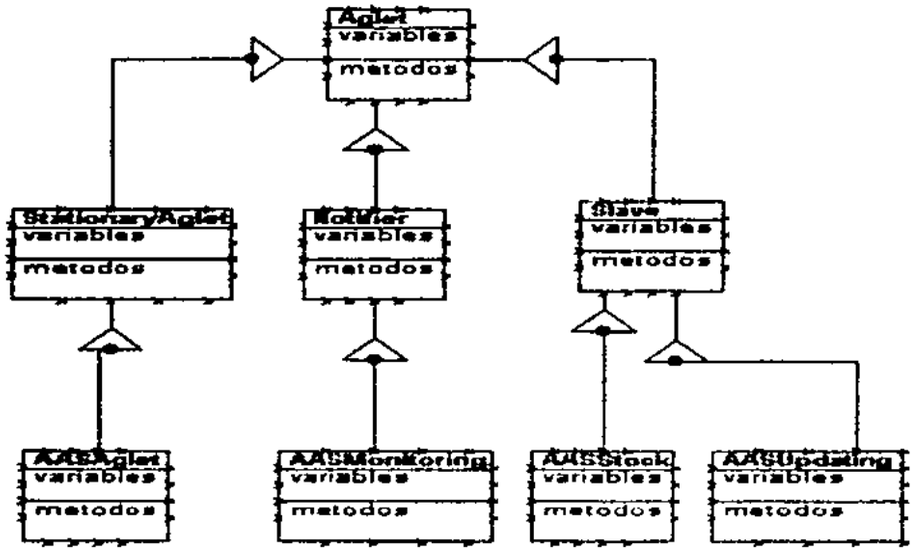
StationaryAglet
Es necesario contar con una clase que nos permita modelar cualquier *aglet* de tipo estacionario. El nombre que mejor representa esta funcionalidad es el de ***StationaryAglet***. Como esta clase debe modelar una funcionalidad de la forma más general posible (con el objeto de ser utilizada en otras aplicaciones) consideramos que debe ser una clase abstracta.

AASAglet
Es una clase concreta cuyo propósito es implementar un objeto estacionario el cual fungirá como administrador de los *aglets* móviles. Este *aglet* administra y envía a los demás *aglets* para realizar sus tareas. Este *aglet* contará con una ayuda gráfica (GUI) para interactuar con el administrador.

AASMonitoring
Las instancias de esta clase nos ayudarán a realizar el monitoreo de archivos en cada una de las máquinas. Debido a que la tarea de monitoreo de archivos se debe hacer de forma periódica y simultánea en cada una de las máquinas a cargo del administrador, se deberá crear una instancia de esta clase por cada máquina que se desee monitorear.

AASStock
Las instancias de esta clase nos ayudarán a recabar la información de cada una de las máquinas. Como la tarea para la recuperación de la información no requiere hacerse de forma simultánea sólo es necesario la creación de una instancia que será la encargada de ir a cada una de las máquinas para cumplir esta tarea.

AASUpdating
Las instancias de esta clase nos ayudarán a la actualización de archivos en cada una de las máquinas que así lo requieran. Como la tarea para la actualización de la información no requiere hacerse de forma simultánea sólo es necesario la creación de una instancia que será la encargada de ir a cada una de las máquinas para cumplir esta tarea.



Jerarquía de clases del paquete unam.as.aglets

Subsistema de la Interfaz gráfica

Este subsistema consta de 6 clases, las primeras cuatro sirven como un medio de comunicación entre el usuario y los *aglets*. Las 2 ultimas sirven como un medio de configurar y obtener información del sistema en general.

AASWindowAglet
Esta clase crea la ventana principal del *aglet* estacionario. Es necesaria para que el administrador pueda estar observando y controlando la actividad de cada uno de los *aglets*.

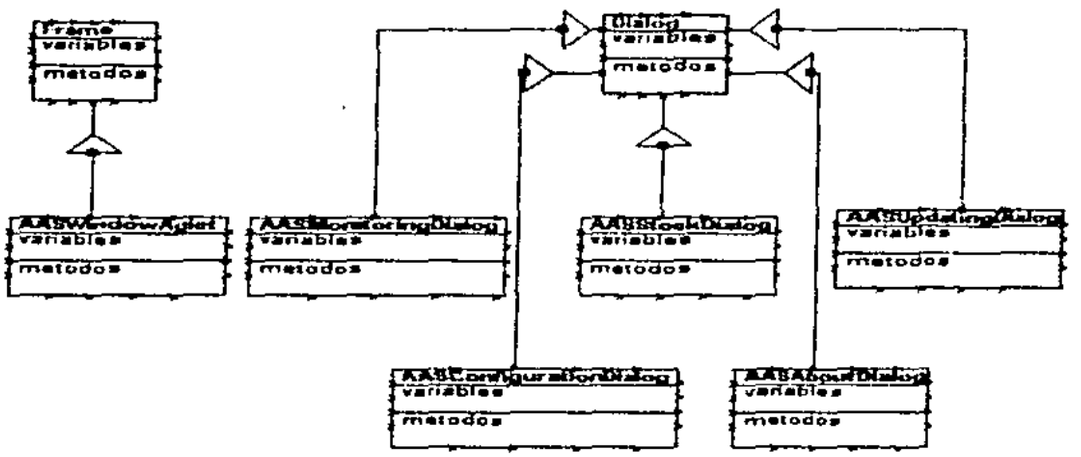
AASMonitoringDialog
Esta clase crea una ventana que permite configurar las características de los *aglets* encargados del monitoreo de archivos.

AASStockDialog
Esta clase crea una ventana que permite configurar las características del *aglet* encargado de la recuperación de información de las máquinas.

AASUpdatingDialog
Esta clase crea una ventana que permite configurar las características del *aglet* encargado de la actualización de la información de las máquinas.

AASConfigurationDialog
Esta clase crea una ventana de diálogo en la cual se permite indicar las rutas (directorios) donde serán guardados los reportes, los archivos de log y los archivos de configuración.

AASAboutDialog
Esta clase crea una ventana que proporciona información relacionada con la versión del sistema y los creadores del mismo.



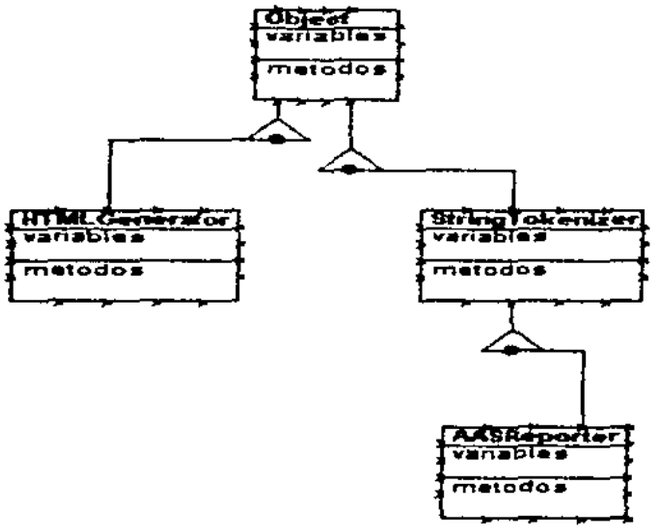
Jerarquía de clases para el paquete unam.iesgui

Subsistema para la creación de reportes

Este subsistema consta de dos clases que se encargan de la generación de los reportes y documentos.

AASReporter
Las instancias de esta clase se encargan de obtener la información de cada uno de los resultados obtenidos por los *aglets* de alguna fuente y proporcionarla al generador de *HTML*, de tal manera que el administrador tenga la información en un formato compacto y portable.

AASHTMLGenerator
Las instancias de esta clase se encargan de la generación de documentos en *HTML* a partir de la información proporcionada por algún otro objeto.



Jerarquía de clases del paquete unam.aas.reporter

Subsistema de utilerías

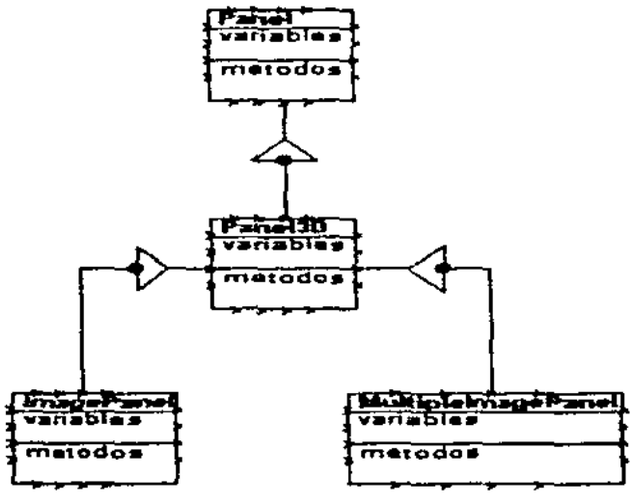
Este subsistema consta de tres clases que se encargan de apoyar la construcción del sistema de la interfaz gráfica.

Panel3D
Las instancias de esta clase se encargan de extender las capacidades de la clase ***Panel*** creando un marco en 3D..

ImagePanel
Las instancias de esta clase se encargan de extender las capacidades de la clase ***Panel3D*** al desplegar y actualizar adecuadamente la imagen que se desea desplegar en el panel.

MultipleImagePanel

Las instancias de esta clase se encargan de extender las capacidades de la clase ***Panel3D*** por medio de desplegar y actualizar múltiples imágenes que se desean desplegar en el panel.



Jerarquía de clases del paquete uniandes.util

Definición de responsabilidades

En esta etapa se describen de manera detallada cada una de las responsabilidades de los objetos involucrados en el sistema.

Subsistema para la creación y manipulación de aglets

StationaryAglet

Los objetos de sus subclases deben conocer:

⇒ La interfaz gráfica a través de la cual se van a comunicar con el usuario.

Los objetos de sus subclases deben hacer las siguientes actividades:

⇒ Inicializar su estado
⇒ Indicar que no pueden desplazarse
⇒ Indicar que no se pueden crear instancias de ***StationaryAglet***
⇒ Proporcionar una respuesta por omisión para cada una de las acciones definidas

AASAglet

Los objetos de esta clase deben conocer:

⇒ Los *aglets* móviles que va a controlar
⇒ Conocer su estado

Los objetos de esta clase deben hacer las siguientes actividades:

⇒ Manejar los mensajes provenientes de los *aglets* móviles
⇒ Realizar las acciones correspondientes dependiendo de los mensajes de los *aglets* móviles.
⇒ Guardar la información que recibe de los *aglets* a su cargo.
⇒ Informar de la localización de esta información a su interfaz gráfica.
⇒ Mandar mensajes a los *aglets* móviles para indicarles alguna acción.
⇒ Controlar la actividad de todos los aglets a su cargo
⇒ Enviar una señal a sus *aglets* para indicarles que se deben autodestruir, cuando esto sea necesario.

AASMonitoring

Los objetos de esta clase deben conocer:

⇒ La máquina a donde van
⇒ Los archivos a monitorear
⇒ Su dueño
⇒ Su tiempo de vida
⇒ El intervalo del monitoreo
⇒ Si regresa o no

Los objetos de esta clase deben hacer las siguientes actividades:

⇒ Realizar el monitoreo en un cierto intervalo de tiempo
⇒ Supervisar el tamaño, los nombres y fecha de acceso a los archivos.
⇒ Notificar inmediatamente a su dueño
⇒ Reportar cualquier error o problema.
⇒ Autodestruirse una vez terminado su tiempo de vida o su dueño se lo solicita explícitamente
⇒ Regresar al servidor si así está especificado o permanecer en la máquina remota
⇒ Comunicarse con otros *aglets* móviles

AASStock

Los objetos de esta clase deben conocer:

⇒ Las máquinas a donde van
⇒ La fuente de información
⇒ Su dueño

Los objetos de esta clase deben hacer las siguientes actividades:

⇒ Regresar y reportar a su dueño las actividades realizadas
⇒ Reportar cualquier error o problema.
⇒ Autodestruirse una vez terminada su tarea o si su dueño se lo solicita explícitamente
⇒ Comunicarse con otros aglets móviles

AASUpdating

Los objetos de esta clase deben conocer:

⇒ Las máquinas a donde van
⇒ La información a actualizar
⇒ Su dueño

Los objetos de esta clase deben hacer las siguientes actividades:

⇒ Regresar y reportar a su dueño las actividades realizadas
⇒ Reportar cualquier error o problema.
⇒ Autodestruirse una vez terminada su tarea o si su dueño se lo solicita explícitamente
⇒ Comunicarse con otros *aglets* móviles

Subsistema de la Interfaz gráfica

AASWindowAglet

Los objetos de esta clase deben conocer:

⇒ Todas las ventanas de diálogo que se utilizarán para configurar tanto el sistema como a cada uno de los *aglets* por separado
⇒ Los componentes que constituyen la interfaz gráfica
⇒ El *aglet* al cual pertenece
⇒ El objeto que genera los reportes

Los objetos de esta clase deben hacer las siguientes actividades:

⇒ Actuar de acuerdo a los eventos recibidos por parte del usuario, abrir las ventanas de diálogo, pedirles o mandarles información a estas
⇒ Debe controlar al objeto que genera los reportes
⇒ Liberar todos los recursos que ella y los diálogos utilicen
⇒ Guardar el estado de configuración al momento de cerrarse
⇒ Recuperar la información de configuración al momento de iniciarse
⇒ Desplegar adecuadamente toda información relacionada con los *aglets*.

AASMonitoringDialog

Los objetos de esta clase deben conocer:

⇒ La lista de máquinas que el usuario desea monitorear
⇒ La lista de archivos que el usuario desea monitorear
⇒ El tiempo de monitoreo
⇒ El intervalo de monitoreo

Los objetos de esta clase deben hacer las siguientes actividades:

⇒ Devolver la información que tiene almacenada
⇒ Actualizar adecuadamente la información que tiene almacenada
⇒ Evitar que su ventana padre pueda seguir operando (modal)
⇒ Permitir agregar o quitar máquinas y archivos de las listas correspondientes
⇒ Permitir configurar otros parámetros
⇒ Diferenciar entre WSs y PCs

AASStockDialog

Los objetos de esta clase deben conocer:

⇒ La lista de máquinas de donde el usuario desea recabar información
⇒ La fuente de información

Los objetos de esta clase deben hacer las siguientes actividades:

⇒ Devolver la información que tiene almacenada
⇒ Actualizar adecuadamente la información que tiene almacenada
⇒ Evitar que su ventana padre pueda seguir operando (modal)
⇒ Permite agregar o quitar máquinas de la lista correspondiente
⇒ Permite configurar la fuente de información
⇒ Diferenciar entre WSs y PCs

AASUpdatingDialog

Los objetos de esta clase deben conocer:

⇒ La lista de máquinas que el usuario desea actualizar
⇒ La lista de archivos a actualizar

Los objetos de esta clase deben hacer las siguientes actividades:

⇒ Devolver la información que tiene almacenada
⇒ Actualizar adecuadamente la información que tiene almacenada
⇒ Evitar que su ventana padre pueda seguir operando (modal)
⇒ Permite agregar o quitar máquinas y archivos de las listas correspondientes
⇒ Permite configurar la fuente de información
⇒ Diferenciar entre WSs y PCs

AASConfigurationDialog

Los objetos de esta clase deben conocer:
- ⇒ La ruta de los archivos de log
- ⇒ La ruta de los archivos de reporte
- ⇒ La ruta de los archivos de configuración

Los objetos de esta clase deben hacer las siguientes actividades:
- ⇒ Devolver la información que tiene almacenada
- ⇒ Actualizar adecuadamente la información que tiene almacenada
- ⇒ Evitar que su ventana padre pueda seguir operando (modal)

AASAboutDialog

Los objetos de esta clase deben conocer:
- ⇒ Información sobre la aplicación y sus creadores

Los objetos de esta clase deben hacer las siguientes actividades:
- ⇒ Mostrar la información adecuadamente
- ⇒ Evitar que su ventana padre pueda seguir operando (modal)

Subsistema para la creación de reportes

AASReporter

Los objetos de esta clase deben conocer:
- ⇒ El lugar de donde se extrae la información
- ⇒ El objeto que crea los reportes

Los objetos de esta clase deben hacer las siguientes actividades:
- ⇒ Crear una fuente de donde se extrae la información
- ⇒ Preprocesar y mandar esta información al generador de documentos en *HTML*
- ⇒ Notificar cualquier error durante el proceso

HTMLGenerator

Los objetos de esta clase deben conocer:
- ⇒ Los tags disponibles para el *HTML*
- ⇒ La ruta donde se guardaran los documentos

Los objetos de esta clase deben hacer las siguientes actividades:
- ⇒ Generar los documentos utilizando *HTML*
- ⇒ Notificar cualquier error durante el proceso

Subsistema de utilerías

Panel3D

Los objetos de esta clase deben conocer:
⇒ El color con el que se pintara el marco

Los objetos de esta clase deben hacer las siguientes actividades:
⇒ Crear un marco en 3D con el color indicado

ImagePanel

Los objetos de esta clase deben conocer:
⇒ La imagen que van a desplegar.
⇒ El color con el que va a pintar el marco

Los objetos de esta clase deben hacer las siguientes actividades:
⇒ Pintar la imagen que se le especifica.
⇒ Actualizarla y repintarla cuando sea necesario.

MutipleImagePanel

Los objetos de esta clase deben conocer:
⇒ Las imágenes que van a desplegar.
⇒ El color con el que va a pintar el marco

Los objetos de esta clase deben hacer las siguientes actividades:
⇒ Pintar las imágenes en el orden que se le especifica.
⇒ Actualizarla y repintarla cuando sea necesario.

Colaboración entre los objetos.

En esta fase se define la forma en que se interrelacionan los objetos dentro del sistema.

Subsistema para la creación y manipulación de aglets

StationaryAglet
Esta clase hereda de *Aglet*. Esto es con el propósito de aprovechar todas las características que debe tener un *aglet*. Pero se caracteriza por evitar el aspecto móvil. Debido a que esta clase es abstracta no colabora directamente con ninguna otra clase. Esta clase proporciona un marco para la creación de *aglets* estacionarios.

AASAglet
Esta clase hereda de ***StationaryAglet*** y debe colaborar con:

⇒ ***AASWindowAglet***. Ésta proporciona el medio de comunicación con el usuario.

⇒ ***AASMonitoring***. ***AASAglet*** debe crear instancias de esta clase para mandarlas a monitorear. Además debe de manejar mensajes que este *aglet* le envíe.

⇒ ***AASStock***. ***AASAglet*** debe crear instancias de esta clase para mandarlas a recabar información en las máquinas remotas. Además debe manejar mensajes que este *aglet* le envíe.

⇒ ***AASUpdating***. ***AASAglet*** debe crear instancias de esta clase para mandarlas a actualizar información en las máquinas remotas. Además debe de manejar mensajes que este *aglet* le envíe.

⇒ ***AgletContext***. ***AASAglet*** debe preguntarle información a este objeto con respecto al entorno en el cual trabaja. Además debe proporcionar este contexto a los *aglets* móviles.

⇒ ***Notifier***. ***AASAglet*** le indica a esta clase que desea crear una instancia de la clase ***AASMonitoring***.

⇒ ***Slave***. ***AASAglet*** le indica a esta clase que desea crear una instancia de la clase ***AASStock*** así como ***AASUpdating***.

⇒ ***Messenger***. ***AASAglet*** crea instancias de esta clase cada vez que necesita enviar algún mensaje a los *aglets* móviles que tiene a su cargo.

AASMonitoring
Esta clase hereda de ***Notifier*** y debe colaborar con:

⇒ ***AASAglet***. ***AASMonitoring*** le indica a las instancias de esta clase las acciones que el usuario desea.

⇒ ***Messenger***. ***AASMonitoring*** recibe y envía mensajes a través de objetos pertenecientes a esta clase.

⇒ ***AgletContext***. ***AASMonitoring*** obtiene información con respecto a su medio de ejecución por medio de este objeto.

AASStock
Esta clase hereda de ***Slave*** y debe colaborar con:

⇒ ***AASAglet***. ***AASStock*** le indica a las instancias de esta clase las acciones que el usuario desea.

⇒ ***Messenger***. ***AASStock*** recibe y envía mensajes a través de objetos pertenecientes a esta clase.

⇒ ***AgletContext***. ***AASStock*** obtiene información con respecto a su medio de ejecución por medio de este objeto.

AASUpdating
Esta clase hereda de ***Slave*** y debe colaborar con:

⇒ ***AASAglet***. ***AASUpdating*** le indica a las instancias de esta clase las acciones que el usuario desea.

⇒ ***Messenger***. ***AASUpdating*** recibe y envía mensajes a través de objetos pertenecientes a esta clase.

⇒ ***AgletContext***. ***AASUpdating*** obtiene información con respecto a su medio de ejecución por medio de este objeto.

Subsistema de la Interfaz gráfica

AASWindowAglet
Esta clase hereda de ***Frame*** y colabora con:

⇒ ***AASMonitoringDialog***. A los objetos de esta clase le pregunta las propiedades de los *aglets* de la clase ***AASMonitoring***.

⇒ ***AASStockDialog***. A los objetos de esta clase le pregunta las propiedades de los *aglets* de la clase ***AASStock***.

⇒ ***AASUpdatingDialog***. A los objetos de esta clase le pregunta las propiedades de los *aglets* de la clase ***AASUpdating***.

⇒ ***AASConfigurationDialog***. A los objetos de esta clase le pregunta la configuración en donde se especifican los directorios de log, reportes y configuración del sistema.

⇒ ***AASAboutDialog***. Le pide que despliegue la información del sistema

⇒ ***AASAglet***. ***AASMonitoring*** le indica a las instancias de esta clase las acciones que el usuario desea.

⇒ ***AASReporter***. A los objetos de esta clase les pide generar los reportes pedidos por el usuario.

AASMonitoringDialog
Esta clase hereda de ***Dialog*** y colabora con:

⇒ ***AASWindowAglet***. ***AASMonitoringDialog*** le proporciona las propiedades de los *aglets* para el monitoreo.

AASStockDialog
Esta clase hereda de ***Dialog*** y colabora con:

⇒ ***AASWindowAglet***. ***AASStockDialog*** le proporciona las propiedades de los *aglets* para el monitoreo.

AASUpdatingDialog
Esta clase hereda de ***Dialog*** y colabora con:

⇒ ***AASWindowAglet***. ***AASUpdating*** le proporciona las propiedades de los *aglets* para el monitoreo.

AASConfigurationDialog
Esta clase hereda de ***Dialog*** y colabora con:

⇒ ***AASWindowAglet***. ***AASConfigurationDialog*** le proporciona la configuración del sistema.

AASAboutDialog
Esta clase hereda de ***Dialog*** y colabora con:

⇒ ***AASWindowAglet***. ***AASAboutDialog*** despliega información del sistema cuando se lo pide.

Subsistema para la creación de reportes

AASReporter
Esta clase hereda de ***Object*** y colabora con:

⇒ ***AASWindowAglet***. ***AASReporter*** coordina la generación de reportes cuando este objeto se lo pide.

⇒ ***HTMLGenerator***. A este objeto le pide crear los documentos *HTML*.

HTMLGenerator
Esta clase hereda de ***Object*** y colabora con:

⇒ ***AASReporter***. Este objeto le proporciona la información y la forma en que desea que sea presentada.

Subsistema de utilerías

Panel3D
Esta clase hereda de ***Panel*** y colabora con:

⇒ ***ImagePanel***.
⇒ ***MultipleImagePanel***.

ImagePanel
Esta clase hereda de ***Panel3D*** y colabora con:

⇒ ***AASAboutDialog***.
⇒ ***AASConfigurationDialog***.
⇒ ***AASMonitoringDialog***.
⇒ ***AASStockDialog***.
⇒ ***AASUpdatingDialog***.

MultipleImagePanel
Esta clase hereda de ***Panel3D*** y colabora con:

⇒ ***AASWindowAglet***.

Implementación

Prototipo

Después de llevar a cabo las fases señaladas el diseñador deberá proceder a la creación de un prototipo, el cual ya en principio tendrá cierta funcionalidad. Esto es hecho con el objetivo de retroalimentar las fases anteriores y si es necesario replantear algunas ideas.

En el presente trabajo los tres subsistemas propuestos quedaron especificados con la creación de tres paquetes independientes: ***unam.aas.aglet***, ***unam.aas.gui*** y ***unam.aas.report*** respectivamente. Los nombres de estos paquetes se basan en el siguiente esquema de acuerdo al estándar que se utiliza por lo general en la creación de paquetes en Java es decir: lugar donde se desarrollo el paquete (institución), nombre del proyecto, y la categoría a la que pertenecen las clases.

A continuación se muestran algunas imágenes de la aplicación y se describe su operación.

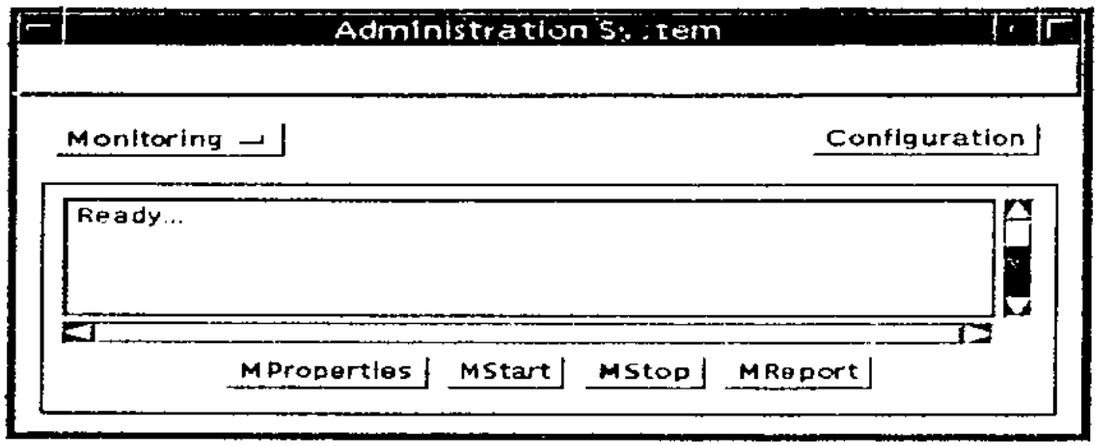


Figura 16 Ventana principal de la aplicación

A través de esta ventana el usuario puede llevar a cabo la administración de los tres tipos de *aglets*: Monitoreo, Actualización e Inventario. La selección del tipo de *aglet* se hace a través del menú de opciones donde aparecen las etiquetas Monitoring, Updating y Stock. En esta etapa de la implementación, la funcionalidad se encuentra limitada al *aglet* de Monitoreo

El botón que tiene la etiqueta "Configuration" sirve para invocar el diálogo desde el cual se capturarán las rutas de los directorios donde se guardara la información necesaria para los tres tipos de *aglets* así como la que estos devuelvan una vez finalizada su tarea.

Al oprimir el botón "Mproperties" obtenemos la ventana de diálogo a través de la cual podremos capturar información necesaria para el *aglet* de Monitoreo.

Para que el *aglet* de Monitoreo comience a realizar su trabajo es necesario oprimir el botón que tiene la etiqueta "MStart"

El botón "MStop" sirve para retraer a los *aglets* que se encuentren realizando la tarea de monitoreo y que por alguna razón deben interrumpirla.

Con el botón "MReport" se inicia la creación de un reporte en formato *HTML* de las actividades hechas por los *aglets* de monitoreo

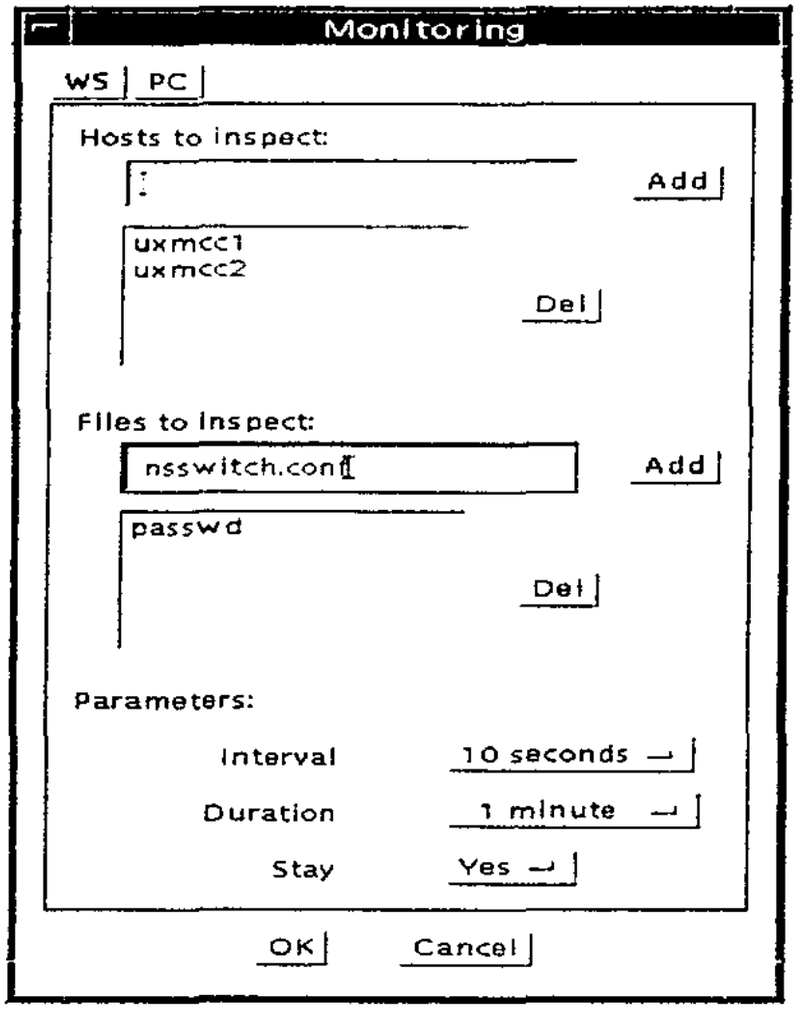


Figura 17 Diálogo para monitoreo de archivos

La ventana de diálogo *Monitoring* es el medio a través del cual el usuario configura a los *aglets* encargados del monitoreo de archivos.

La ventana cuenta con dos botones ("WS" y "PC") los cuales nos sirven para hacer distinción entre estaciones de trabajo y PCs. Esto se debe principalmente a que los sistemas de archivos se trabajan de distinta manera en dichas arquitecturas.

El primer campo de texto sirve para capturar nombres de máquinas en las que posiblemente se desea hacer un monitoreo de los archivos de configuración. Una vez capturado el nombre de la máquina es necesario oprimir el botón "Add" para que el nombre sea agregado a la lista de máquinas a inspeccionar. La lista es una lista de selección múltiple, esta característica nos permite indicar

cuales son las maquinas que queremos monitorear o, si es el caso, para eliminar los nombres de aquellas máquinas que no queremos se encuentren en la lista. El botón "Del", que se encuentra a la derecha de esta lista, nos permite realizar esa operación.

El segundo campo de texto nos sirve para capturar los nombres de los archivos de configuración que posiblemente se quiere monitorear. Para agregar los nombres a la lista es necesario oprimir el botón "Add" que se encuentra a la derecha de la lista de archivos a inspeccionar. Esta lista tiene una funcionalidad semejante a la lista de máquinas a inspeccionar.

La sección de "Parámetros" contiene tres menús de selección a través de los cuales le especificaremos a los *aglets* el tiempo que permanecerán monitoreando los archivos en la máquina asignada, el cual va de 1 minuto a un día, el intervalo de tiempo que debe transcurrir entre cada monitoreo ("10 segundos", "30 segundos", "1 min", etc.) y si debe regresar o quedarse en la máquina, una vez que ha notado cambios en el archivo o archivos inspeccionados.

Para aceptar los cambios hechos en esta ventana de diálogo basta con oprimir el botón "OK", o si no se aceptan y se prefiere mantener la configuración anterior es necesario presionar el botón "Cancel".

Extensión del Prototipo

Una vez realizado el prototipo y verificado que trabaja de acuerdo a las especificaciones establecidas por el diseñador éste puede decidir ampliar el prototipo para que cuente con una funcionalidad completa.

En el proyecto propuesto como parte central de esta tesis se optó por presentar el prototipo con una funcionalidad completa.

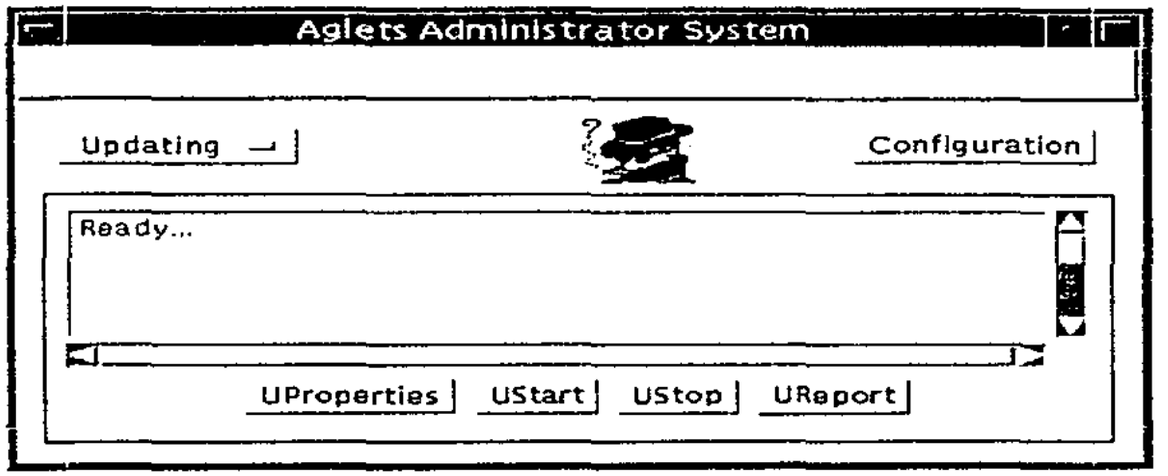


Figura 18 Ventana principal con funcionalidad completa

En la extensión del prototipo se agregó la funcionalidad del menú de selección a la ventana principal para poder invocar a las ventanas de diálogo correspondientes a las propiedades de los *aglets* de Actualización y de Inventario. Además se hizo distinción de las tres consolas con la ayuda de un marco de color: negro para la actividad de monitoreo, azul para la actividad de actualización y rojo para la de inventario. En la barra de menú se habilitó la opción de "Help", la cual nos proporciona información con respecto al sistema. La funcionalidad de cada una de las consolas es similar a la descrita en la fase inicial del prototipo.

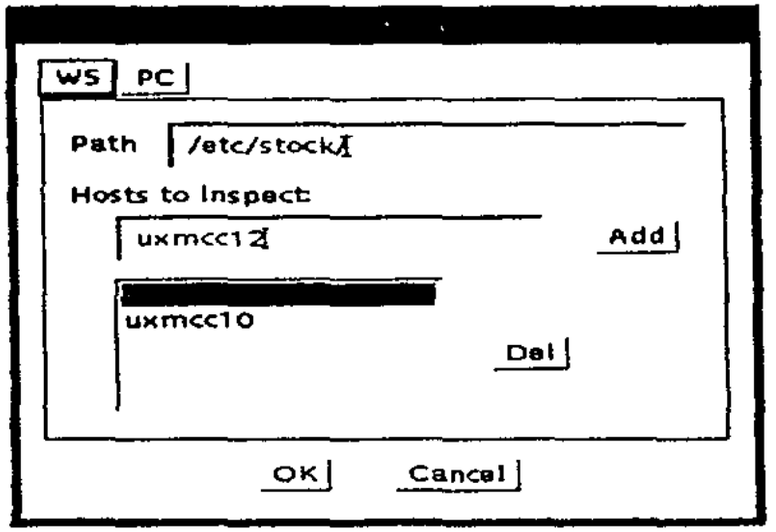


Figura 19 Diálogo para inventario

La ventana de diálogo correspondiente a las propiedades del *aglet* de Inventario nos permite seleccionar la plataforma en la cual va a trabajar el agente. A través de un campo de texto se le indica el directorio donde se encuentran los archivos de inventario y con la ayuda de otro campo de texto y una lista de selección múltiple se agregan las máquinas de las cuales se desea obtener la información correspondiente al inventario.

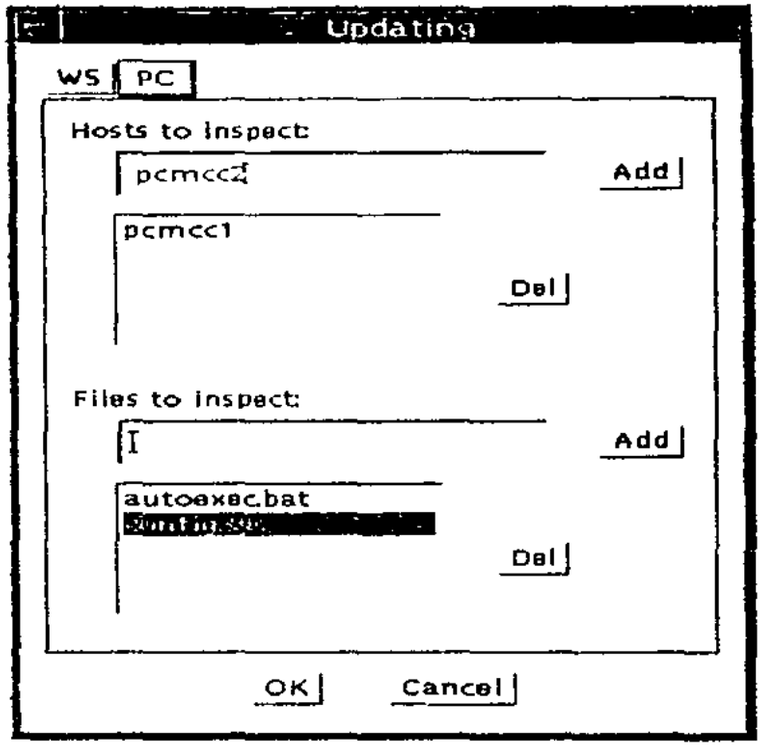


Figura 20 Diálogo para actualización de archivos

Este diálogo nos permite especificar las máquinas donde se desea hacer la actualización de los archivos de configuración seleccionados en la lista correspondiente.

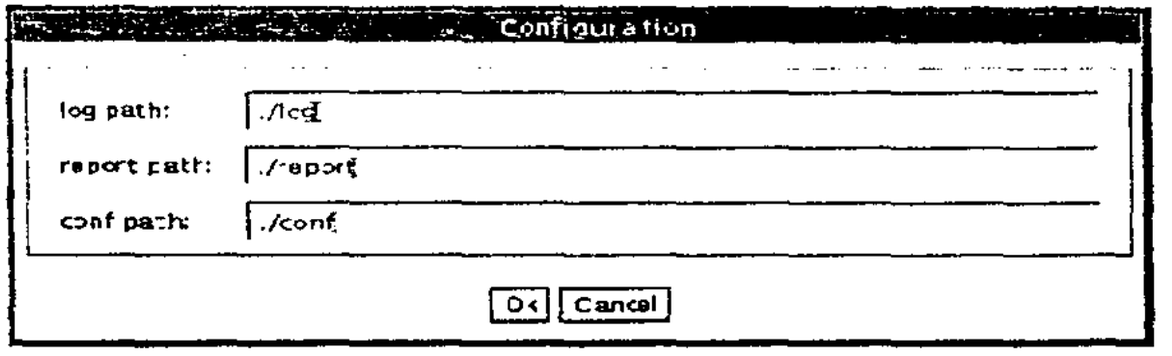


Figura 21 Diálogo para configuración de la aplicación

En el diálogo de configuración del sistema se hace la captura de las rutas de los directorios donde se guardarán los archivos de log, de reportes y el directorio de donde deberá leer sus archivos de configuración.

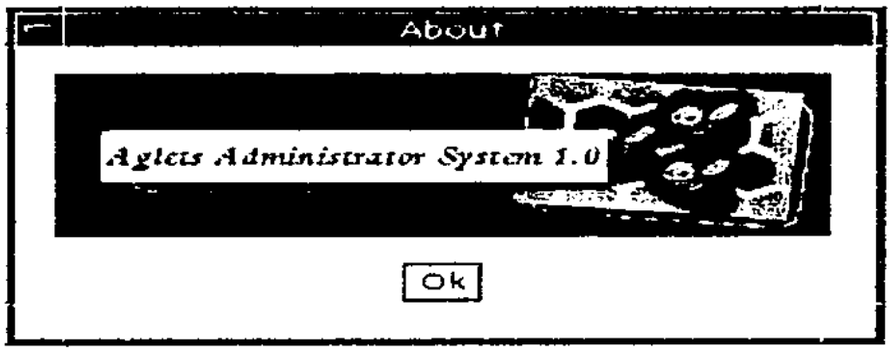


Figura 22 Diálogo de información de la aplicación

Este diálgo solo muestra información con respecto a la versión del sistema.

Pruebas

Se realizaron diferentes tipos de pruebas para comprobar que la funcionalidad del sistema es la deseada. Dentro de las pruebas que se hicieron están:

- Ver el comportamiento de los *aglets* cuando la conexión de la red se pierde.
- Confirmar que los *aglets* hacen el trabajo que se les pide.
- Confirmar que los *aglets* hacen el trabajo en el orden que se les pide.
- Verificar que los *aglets* reporten adecuadamente sus actividades.
- Medir los recursos consumidos por el sistema y cada una de sus partes.

Capítulo 6

Resultados y Conclusiones

En este capítulo se presentan los resultados obtenidos de las pruebas realizadas al sistema en operación, también se expone un análisis de estos resultados y del trabajo en general.

Presentación de resultados

La siguiente tabla muestra los recursos que cada una de las partes del sistema demanda.

programa	tamaño en bytes	Uso de CPU	Tiempo de vida
máquina virtual	2877611	*	**
servidor *aglets*	352884	*	**
agente de monitoreo	1828	(2*TT)+(NI*TA)	(2*TT)+(NI*TA) +TD
agente de inventario	1263 + datos	((2*TT)+TA) * NM	((2*TT)+TA) * NM
agente de actualización	1050 + archivos	((2*TT)+TA) * NM	((2*TT)+TA) * NM
agente de mensajes	2776	TT+TA	TT+TA

* Solo cuando existe un agente activo o se recibe una petición de mandar o recibir un agente.
** En memoria todo el tiempo que el programa se ejecuta
TT: Tiempo de Transferencia
NI: Número de Intervalos en los que se realiza la tarea
TA: Tiempo en el que realiza su Actividad
NM: Número de Máquinas visitadas
TD: Tiempo en el que el agente permanece Dormido

Como se puede observar en el primer renglón, la máquina virtual es la que demanda mas recursos tanto de memoria como del procesador. Esta parte siempre esta presente en cualquier aplicación hecha con Java. La segunda parte que consume más recursos es el servidor que esta en espera de nuevos *aglets* y que se encarga de administrar los que ya se encuentran activos en la máquina local. Los últimos cuatro agentes son los agentes móviles, y, como se puede ver, son bastante pequeños. Aquí es donde se ve la ventaja de utilizar agentes: puesto que la cantidad de información transmitida por la red es mínima. Esto se puede ver comparando el tiempo de vida del *aglet* con el tiempo en que el *aglet* es transferido por la red (TT).

Por otra parte todas las fórmulas muestran que si la actividad del *aglet* aumenta considerablemente, altera muy poco el tiempo de uso de la red (en los casos de

los *aglets* de actualización y de inventario) o no lo cambia absolutamente en nada (en el caso del *aglet* de monitoreo).

Los *aglets* realizaron adecuadamente sus labores en el orden indicado y reportando oportunamente sus actividades. Algo que se observo que es muy importante es el tener adecuadamente la hora y fecha de cada una de las máquinas para que los reportes no muestren datos erróneos.

Actualmente el sistema contempla la posibilidad de que la red no esté disponible cuando el agente necesite desplazarse, sin embargo no existe forma de recuperar el estado de un agente si un cliente o el servidor son dados de baja de forma inesperada.

Una vez analizados los datos, procedemos a hacer algunas observaciones con respecto al desarrollo del trabajo. Esta parte se encuentra dividida en las siguientes secciones:

- Esquema de desarrollo
- Agentes
- Java
- Framework
- Ambientes de desarrollo
- Seguridad
- Tecnologías usadas
- Internet
- Proyección comercial

Iniciamos con el análisis del esquema utilizado en la creación del sistema.

Esquema de desarrollo

Ventajas:

- En cuanto a las ventajas del esquema de desarrollo utilizado en la implementación del sistema cabe señalar que los cambios a los *aglets* no afectan el diseño principal de la aplicación, por lo que éstos no repercuten en otras partes del sistema. Además, el sistema permite la inserción de nuevos *aglets* para aumentar la funcionalidad del sistema.

- Debido a que en la actualidad no existe una metodología orientada a objetos estándar de análisis y de diseño para el desarrollo de aplicaciones, y a que varias de las metodologías existentes en ocasiones son densas y difíciles de aplicar, decidimos basarnos en el desarrollo de un prototipo rápido con cierta funcionalidad para posteriormente extenderlo. Para ello aplicamos algunos de los conceptos del Diseño Orientado a Objetos de Rebecca Wirfs-brock (diseño basado en responsabilidades), del capítulo 11 del manual de programación de Smalltalk/V (diseño de un prototipo), y de la simbología para diagramas de clases de Peter Coad, con el fin de tener un marco de referencia para el desarrollo de la aplicación. Esta decisión fue tomada con

base en la experiencia obtenida al desarrollar otras aplicaciones. Lo anterior nos permitió tener resultados de manera casi inmediata. Esto es muy importante debido a que nos dio un panorama general del funcionamiento del sistema y de esa manera poderlo extender fácilmente. Esta forma de crear la aplicación nos permitió además retroalimentar las fases de análisis y diseño y hacer cambios en la creación de subsistemas y de clases, con el objetivo de lograr mayor modularidad y consistencia en la aplicación

Desventajas:

- Al no utilizar una metodología orientada a objetos en particular, sino que más bien se utilizaron conceptos de varias de las metodologías conocidas, no se contó con una herramienta que nos facilitara la documentación para las clases y objetos de nuestra aplicación.

- Dado que el diseño interno de los *aglets* que trabajan dentro del sistema depende fuertemente del framework proporcionado por IBM, si éste sufriera modificaciones substanciales en próximas versiones, como fue el paso de la versión *alpha3* a la *alpha4*, nuestros *aglets* deberán ser modificados para que funcionen correctamente. Esta dependencia se debe principalmente a la forma en que los *aglets* se comunican entre sí y su contexto de ejecución.

  Posibles soluciones: implementar un framework propio para el transporte y manipulación de los agentes o esperar a que se libere una versión ya comercial del framework.

Agentes

Ventajas:

- Es una tecnología que está creciendo rápidamente.

- Existen varias instituciones, tanto educativas como comerciales realizando investigaciones en esta área.

- Es una tecnología que puede ser aplicada en una gran cantidad de campos y problemas.

- Proveen un medio para aprovechar mejor los recursos disponibles en una red, así como la red misma.

Desventajas:

- Existe poca información con respecto a esta tecnología. Sólo existen unos pocos libros, pero por lo regular son recopilaciones de artículos.

- No se ha llegado a un acuerdo claro de lo que involucra esta tecnología. ¿Cómo dividir a los tipos de agentes?, ¿qué son?, etc.

- Son muy pocas las aplicaciones a nivel comercial que utilizan esta tecnología, por lo que hay que esperar para saber como será aceptada.

Java

Ventajas:

- Con la misma versión del JDK el sistema funciona perfectamente en diferentes plataformas sin cambio alguno. Dado que Java es un lenguaje orientado a objetos el análisis y diseño que realizamos fue simple de implementar. Su concepto de paquetes ayuda a la modularidad del sistema y a evitar dependencias innecesarias.

- El lenguaje está mostrando un gran avance en el área de criptografía, lo cual permitirá aumentar considerablemente la seguridad en nuestro sistema.

- Una vez liberada la versión final del JDK1.1 nuestro sistema podrá ser ejecutado en cualquier plataforma que soporte esta versión.

Desventajas:

- Los paquetes ***RMI*** y ***ObjectSerialization*** sólo están disponibles actualmente en equipo SUN y PC, por lo que nuestro sistema está actualmente limitado a estas plataformas.

  Posibles soluciones: La liberación del JDK1.1 por parte de SUN incluirá los paquetes de ***RMI*** y ***OS*** como parte del lenguaje y estará disponible para todas las plataformas.

- Problemas de compatibilidad entre diferentes maquinas virtuales. Éste fue uno de los principales problemas con el que nos enfrentamos durante el desarrollo de la aplicación. Haciendo notable la inmadurez del lenguaje y la existencia de diferentes implementaciones no compatibles que ya existen en el mercado y que luchan por ganar éste. Un ejemplo de lo anterior se presento al tratar de probar nuestra aplicación o el *Tahiti* con diferentes versiones de la VM, incluso realizadas por la misma compañía. Al hacer esto descubrimos que no funcionaban adecuadamente o que el sistema simplemente abortaba.

  Posibles soluciones: Por el momento sólo queda utilizar versiones no optimizadas de la maquina virtual hasta que no se defina todo lo que pasará a ser parte del lenguaje.

- Incompatibilidad entre diferentes paquetes y sus versiones. Un problema similar al anterior fue la incompatibilidad entre las distintas versiones de Java, RMI, Object Serialization, y AWB; ya que algunas versiones de cada uno de ellos no funcionan con un sistema pero con otros si.

Framework

Ventajas:

- La elección del Aglets Workbench como base central para el desarrollo de nuestra aplicación fue la más indicada; ya que ha mostrado ser el framework más completo y el que se encuentra en constante actualización, en constraste notable con otros frameworks similares.

- El framework utilizado nos permitió crear varios administradores de aglets sobre una misma máquina de tal manera que durante el desarrollo del sistema realizamos todas las pruebas con los aglets sin necesidad de utilizar un servidor remoto, lo cual represento un ahorro considerable en la etapa de desarrollo. Además, nos permitió configurar una gran cantidad de parámetros relativos a la seguridad y otros referentes a la administración de la red.

- La interfaz gráfica incluida en el framework nos permitió a su vez el crear, administrar y destruir *aglets* de una forma simple y consistente durante todo el desarrollo. Esta interfaz nos proporciono el contexto sobre el que corren todos los *aglets* utilizados en el sistema, lo cual nos facilitó el desarrollo de éste.

- Otra característica que es conveniente señalar es la referente a que los *aglets* creados y compilados en el sistema operativo de SUN funcionan exactamente igual en el sistema operativo Windows 95.

Desventajas:

- Al momento de escribir este trabajo IBM no indicaba de que forma va a licenciar el uso de su framework para fines comerciales, lo cual resulta fundamental si se desea comercializar este sistema.

- Por otra parte el pensar en desarrollar un framework propio representaria el dedicar al menos un año de trabajo de cinco o más personas para lograr una funcionalidad equivalente.

- Como el framework utilizado es una versión *alfa*, todavia está sufriendo cambios entre una versión y otra. Algunos de estos cambios afectaron el desarrollo de este trabajo y valdrá la pena comentarlos a continuación:

  Versión alpha3

  ⇒ No permite la creación de subprocesos o la ejecución de comandos del sistema operativo, lo cual es fundamental para un sistema de la naturaleza como el desarrollado; ya que en la implementación requerimos de ello para poder accesar a los recursos de la máquina tanto para la administración como para la utilización de otras herramientas de apoyo por ejemplo: Netscape, para la visualización de los reportes.

⇒ Tampoco da soporte para autentificar y cifrar los *aglets* que viajan y llegan a otro nodo en la red.

⇒ No permite tener el código de los *aglets* que uno crea en un directorio diferente al que se encuentra el código del framework.

⇒ El manejo de eventos en las ventanas muestra algunas inconsistencias, ya que cuando la ventana del *aglet* es creada ésta pertenece a un *thread* de ejecución diferente al del *aglet*, por lo que en ocasiones al tratar de desplegar un diálogo el sistema manda un error.

⇒ No permite la actualización de *aglets* de forma dinámica. Si el código de un *aglet* es modificado y se desea que el ambiente tome en cuenta estos cambios al momento de crear el *aglet* desde la interfaz gráfica es necesario dar de baja al servidor de *aglets* y volver a iniciarlo cada vez que se han hecho dichos cambios.

⇒ No proporciona una forma de habilitar o deshabilitar al administrador de seguridad o de configurarlo desde la interfaz gráfica.

Versión alpha4

⇒ Cambiaron el esquema de pizarrón por el de *multicast*, afectando la forma en que los *aglets* se comunican entre si y con el administrador. Esto derivo en cambios substanciales en nuestros *aglets*.

⇒ Algunas clases y métodos del framework cambiaron de nombre o desaparecieron, por lo que si uno utilizo algunas de ellas tendrá la obligación de revisar profundamente su diseño a fin de evitar el uso de éstas y reemplazarlas por su sustituto.

⇒ La documentación para la programación de los *aglets* es muy limitada, ya que con lo único que se contó fue con la documentación del API, con el código fuente, y con los dos primeros capítulos de un libro escrito por los autores del framework.

⇒ Actualmente no se cuenta con la versión en applet del manejador de aglets, cuyo nombre es *Fiji*. Esto no nos permite ejecutar nuestra aplicación desde un visualizador como Netscape o Internet Explorer. Lo cual implica ocupar espacio en disco en cada una de las máquinas donde se desea ejecutar un servidor de *aglets*.

Ambientes de desarrollo y constructores de interfaces gráficas

En la implementación del sistema fue necesario probar y evaluar diferentes ambientes de desarrollo para la creación de la interfaz gráfica, llegando a la conclusión de no utilizar ninguno de ellos. Los problemas que muestran la mayoría de los ambientes de desarrollo evaluados son:

- Difíciles de utilizar; ya que no cuentan con todos los componentes disponibles en el API de Java.
- El código que generan es muy difícil de modificar manualmente o generan mucha "basura".
- Es necesario contar siempre con la herramienta indicada para hacer modificaciones a la interfaz, lo cual puede ser muy costoso.
- Por otra parte, si el ambiente de desarrollo no está disponible en todas las plataformas con las que puede contar un programador, las posibles extensiones del sistema estarán limitadas a solamente ser llevadas a cabo en aquellas plataformas compatibles con la herramienta de desarrollo.

Seguridad

Ventajas:

- El lenguaje Java por si mismo proporciona ciertos niveles de seguridad para la ejecución del código de los *aglets*, tales como: El ***ClassLoader*** y el ***SecurityManager***.
- El primero nos permitió garantizar que el código de los *aglets* no iba a producir algún daño al servidor o a algún otro *aglet*, mientras que el segundo nos permitió definir políticas propias de seguridad con respecto a lo que un *aglet* puede o no puede hacer en la máquina por ejemplo: leer archivos, borrar archivos, etc.

Desventajas:

- En este momento nuestro sistema no proporciona ningún mecanismo de autentificación y cifrado de la información que transportan los *aglets* ni de los *aglets* mismos.

  Posibles soluciones: Con el surgimiento de la versión 1.1 del JDK de Java, será posible agregar esta funcionalidad a nuestro sistema sin ningún cambio en su estructura.

- Es muy importante tomar en cuenta que la seguridad también está estrechamente relacionada con el sistema operativo sobre el cual se esté trabajando. El caso más notorio es en las PCs con Windows95, donde

cualquier usuario tiene la capacidad de dar de baja el servidor de *aglets*, o incluso dar de alta alguno.

- Como se mencionó, no existe forma de saber si un *aglet* fue enviado por el administrador o por otra persona.

  Posibles soluciones: Sería adecuado definir tanto niveles como áreas de acceso. Lo cual implica restringir el origen de los *aglets* a máquinas locales o incluso a una sola, la cual sólo es accesada por el administrador del sistema

Diferentes tecnologías utilizadas en la creación del sistema

La creación de este sistema nos permitió conocer y estudiar una gran diversidad de tecnologías, con el fin de utilizar el framework y/o implementarlas de forma conjunta para la solución del problema planteado. Entre estas tecnologías se encuentran:

- Procesamiento distribuido. Éste es el concepto básico sobre el cual trabaja nuestro sistema; ya que tiene la capacidad de distribuir el trabajo en todas las maquinas administradas.
- Concurrencia. Ésta es necesaria para que trabajen adecuadamente los agentes y puedan convivir en un mismo ambiente.
- Arquitecturas de pizarrón. Ésta fue necesaria para la comunicación entre agentes hasta la versión alpha3 del framework.
- Multicast. A partir de la versión alpha4, éste es el medio de comunicación entre los agentes.
- Análisis, Diseño y Programación Orientada a Objetos. Útiles para la mejor abstracción de los agentes e implementación de los conceptos de agentes.
- Redes. Necesario para el concepto de agentes móviles y el estudio de protocolos de comunicación.
- Teoría de agentes. Otra de las partes fundamentales sobre las que se sustenta la aplicación.
- Frameworks. Esencial para poder aplicar el AWB como base de nuestro sistema.
- Criptografía. Para conocer posibles formas de garantizar la integridad y autenticidad de la información que maneja el sistema.
- Patterns. Base para el análisis y diseño de los diferentes tipos de agentes utilizados en el sistema.

Internet como un medio para trabajar.

El tener acceso a Internet nos permitió conocer y trabajar con personas que están realizando investigación de frontera en el tema central de este trabajo y que están desarrollando sistemas comerciales y de investigación en las áreas antes citadas. Más aun, nos permitió colaborar directamente con ellos y aportar ideas y sugerencias para el futuro desarrollo del AWB. Esta forma de trabajar resulta muy agradable y productiva; ya que tanto dudas como aportaciones se pueden realizar de una forma muy simple y rápida. Cabe señalar que nuestro principal contacto fue con Mitsuru Oshima, quien trabaja en el Tokio Research Laboratory de IBM, en Japón.

Proyección comercial del sistema

El sistema tiene la capacidad de extenderse para agregar mayor funcionalidad y pueda ser un producto comercializable. Dentro de las extensiones que se le tienen que hacer es proporcionarle un mecanismo sólido de autentificación tanto para los agentes como para la persona que hace uso del sistema. Algo que también se tiene que tomar en cuenta es la disponibilidad y licencias que establezca IBM del uso de su framework con fines comerciales.

Apéndice A

API del Aglets Workbench

Algunas de las clases e interfaces más importantes se muestran a continuación junto con sus respectivos métodos:

Interface aglet.AgletContext

public interface AgletContext
extends Object

The AgletContext class is the execution context for running aglets. It provides means for maintaining and managing running aglets in an environment where the aglets are protected from each other and the host system is secured against malicious aglets.

Methods

- **getAgletProxies**

public abstract Enumeration getAgletProxies()

Gets the aglet proxies in the current execution context.
Returns:
an enumeration of aglet proxies.

- **getAgletProxy**

public abstract AgletProxy getAgletProxy(AgletIdentifier identity)

Gets the proxy for an aglet specified by its identity.
Parameters:
identity - the identity of the aglet.
Returns:
the aglet proxy.

- **getAgletProxy**

public abstract AgletProxy getAgletProxy(URL remote,
AgletIdentifier identity)

Gets the proxy for an remote aglet given by the URL. Please remind that this is tentative API.
Parameters:
identity - the identity of the aglet
Returns:
the aglet proxy

- **getProperty**

```
public abstract Object getProperty(String key)
```

Gets the context property indicated by the key.
Parameters:
key - the name of the context property.
Returns:
the value of the specified key.

- **getProperty**

```
public abstract Object getProperty(String key,
                                   Object def)
```

Gets the context property indicated by the key and default value.
Parameters:
key - the name of the context property.
def - the value to use if this property is not set.
Returns:
the value of the specified key.

- **setProperty**

```
public abstract void setProperty(String key,
                                 Object value)
```

Sets the context property indicated by the key and value.
Parameters:
key - the name of the context property.
value - the value to be stored.
Returns:
the value of the specified key.

- **multicastMessage**

```
public abstract void multicastMessage(Message msg)
```

Sends a multicast message to the subscribers in the context
Parameters:
message - to send
Returns:
void at this moment.

- **createAglet**

```
public abstract AgletProxy createAglet(URL url,
                                       String name,
                                       Object init) throws IOException, AgletException
```

Creates an instance of the specified aglet located at the specified URL.

Parameters:
url - the URL to load the aglet class from.
name - the aglet's class name.
init - the value passed to Aglet.onCreation method as an argument.
Returns:
a newly instantiated and initialized Aglet.
Throws: UnknownHostException
if the given host could not be found.
Throws: ServerNotFoundException
if the server is not found.
Throws: AgletClassNotFoundException
if the given aglet class could not be found.
Throws: AgletInstantiationException
if the instantiation failed.

● **retractAglet**

public abstract AgletProxy retractAglet(URL url) throws IOException, AgletException

Retracts the Aglet specified by its url:
atp://host-domain-name/[user-name]#aglet-identity.
Parameters:
url - the location and aglet identity of the aglet to be
retracted.
Returns:
the aglet proxy for the retracted aglet.
Throws: UnknownHostException
if the specified HOST is not found.
Throws: ServerNotAvailableException
if the aglet server specified in the URL is not available.
Throws: MalformedURLException
if the given url is not URI for an aglet.
Throws: RequestRefusedException
if the retraction refused.
Throws: AgletNotFoundException
if the aglet could not be found.

● **activateAglet**

public abstract AgletProxy activateAglet(AgletIdentifier aid) throws AgletException

Activate an aglet. This is a forced activation of a deactivated aglet.
Returns:
an aglet proxy of the activated aglet.
Throws: AgletNotFoundException
if the aglet could not be found.

● **getHostingURL**

public abstract URL getHostingURL()

Returns the URL of the daemon serving all current execution contexts.

Returns:
the URL of the daemon. null if the Hosting information is not available due to some reason like as security violation.

- **showDocument**

public abstract void showDocument(URL url)

Shows a new document. This may be ignored by the aglet context.
Parameters:
url - an url to be shown

- **getImage**

public abstract Image getImage(URL image)

Gets an image

- **getAudioClip**

public abstract AudioClip getAudioClip(URL audio)

Gets an audio clip

- **getImageData**

public abstract ImageData getImageData(URL image)

Gets an image data. This is a temporary solution.

- **getImage**

public abstract Image getImage(ImageData image)

Gets an image. This is a temporary solution.

Class aglet.Aglet

public class Aglet
extends Object
implements Cloneable, Serializable

The Aglet class is the abstract base class for aglets. Use this class to create your own personalized aglets.

Variables

- **MAJOR_VERSION**

public final static short MAJOR_VERSION

MINOR_VERSION

public final static short MINOR_VERSION

Constructors

Aglet

protected Aglet()

Constructs an uninitialized aglet. This method is called only once in the life cycle of an aglet. As a rule, you should never override this constructor. Instead, you should override onCreation() to initialize the aglet upon creation.

Methods

clone

public final Object clone() throws CloneNotSupportedException

Clones the aglet and the proxy that holds the aglet. Notice that it is the cloned aglet proxy which is returned by this method.
Returns:
the cloned proxy.
Throws: CloneNotSupportedException
when the cloning fails.
Overrides:
clone in class Object

dispatch

public final void dispatch(URL destination) throws IOException, AgletException

Dispatches the aglet to the location (host) specified by the destination argument.
Parameters:
destination - dispatch destination.
Throws: ServerNotAvailableException
if the server could not be found.
Throws: UnknownHostException
if the host given in the URL does not exist.
Throws: RequestRefusedException
if the remote server refused the dispatch request.
Throws: AgletException
if the dispatch request failed.
Throws: InvalidAgletException
if the aglet is not valid.

- **dispose**

public final void dispose() throws InvalidAgletException

Destroys and removes the aglet from its current aglet context. A successful invocation of this method will kill all threads created by the given aglet.
Throws: InvalidAgletException
if the aglet is not valid.

- **deactivate**

public final void deactivate(long duration) throws InvalidAgletException

Deactivates the aglet. The aglet will temporarily be stopped and removed from its current context. It will return to the context and resume execution after the specified period has elapsed.
Parameters:
duration - deactivation period specified in milliseconds
Throws: InvalidAgletException
if the aglet is not valid.
Throws: IllegalArgumentException
if the argument is negative.

- **run**

public void run()

Is the entry point for the aglet's own thread of execution. This method is invoked upon a successful creation, dispatch, retraction, or activation of the aglet.

- **onCreation**

public void onCreation(Object init)

Initializes the new aglet. This method is called only once in the life cycle of an aglet. Override this method for custom initialization of the aglet.
Parameters:
init - the argument with which the aglet is initialized.

- **onClone**

public void onClone()

Initializes the cloned aglet. Override this method for custom initialization of the cloned aglet.

- **onArrival**

public void onArrival()

Initializes the newly arrived aglet. Subclasses may override this method to implement actions that should be taken on arrival of the aglet at a new host.

- **onActivation**

public void onActivation()

Initializes the newly activated aglet. Subclasses may override this method to implement actions that should be taken on activation of the aglet.

- **onCloning**

public void onCloning()

Is called when you attempt to clone an aglet. The default implementation of this method throws a SecurityException to avoid accidental cloning. Subclasses may override this method to implement any actions that should be taken in response to a cloning request.

- **onDispatching**

public void onDispatching(URL destination)

Is called when an attempt is made to dispatch the aglet. Subclasses may override this method to implement actions that should be taken in response to a dispatch request. For example, if you want to create an immobile (stationary) aglet, override this method to throw a SecurityException.
Parameters:
destination - the destination of the dispatch request.
Throws: SecurityException
if the dispatch request is rejected.

- **onDisposing**

public void onDisposing()

Is called when an attempt is made to dispose of the aglet. Subclasses may override this method to implement actions that should be taken in response to a request for disposal.
Throws: SecurityException
if the request for disposal is rejected.

- **onReverting**

public void onReverting(URL remoteURL)

Is called when someone attempts to retract the aglet from the remote location specified by the remoteURL parameter. Subclasses may override this method to implement actions that should be taken in response to a request for retraction. Override this method to throw a SecurityException if you want to deny a request for retraction.

Parameters:
remoteURL - source of retraction request.
Throws: SecurityException
if the request for reverting is rejected.

● **onDeactivating**

public void onDeactivating(long duration)

Is called when an attempt is made to deactivate the aglet. Subclasses may override this method to implement actions that should be taken in response to a request for deactivation.
Parameters:
duration - deactivation period specified in milliseconds

● **handleMessage**

public boolean handleMessage(Message message)

Handles the message form outside.
Parameters:
msg - the message sent to the aglet
Returns:
true if the message was handled. Returns false if the message was not handled. If false is returned, the MessageNotHandled exception is thrown in the FutureReply.getReply and AgletProxy.sendMessage methods.

● **getAgletContext**

public final AgletContext getAgletContext() throws InvalidAgletException

Gets the context in which the aglet is currently executing.
Returns:
the current execution context.
Throws: InvalidAgletException
if the aglet is not valid.

● **getMessageManager**

public final MessageManager getMessageManager() throws InvalidAgletException

Gets the message manager.
Returns:
the message manager.
Throws: InvalidAgletException
if the aglet is not valid.

● **getIdentifier**

public final AgletIdentifier getIdentifier() throws InvalidAgletException

Returns the identity of this aglet.

Throws: InvalidAgletException
if the aglet is not valid.

- **getItinerary**

public final Itinerary getItinerary() throws InvalidAgletException

Returns:
the itinerary Object of the aglet; null otherwise.
Throws: InvalidAgletException
if the aglet is not valid.

- **getCodeBase**

public final URL getCodeBase() throws InvalidAgletException

Gets the code base URL of this aglet
Throws: InvalidAgletException
if the aglet is not valid.

- **getText**

public final String getText() throws InvalidAgletException

Gets the message line of this Aglet.
Throws: InvalidAgletException
if the aglet is not valid.

- **setText**

public final void setText(String text)

Sets the text of this Aglet. A way for the aglet to display messages without opening a window.
Parameters:
message - the message.

- **getProperty**

public final String getProperty(String key) throws InvalidAgletException

Gets the aglet property indicated by the key.
Parameters:
key - the name of the aglet property.
Returns:
the value of the specified key.
Throws: InvalidAgletException
if the aglet is not valid.

- **getProperty**

public final String getProperty(String key,
String defValue) throws InvalidAgletException

Gets the aglet property indicated by the key and default value.
Parameters:
key - the name of the aglet property.
defValue - the default value to use if this property is not set.
Returns:
the value of the specified key.
Throws: InvalidAgletException
if the aglet is not valid.

● **setProperty**

public final void setProperty(String key,
String value) throws InvalidAgletException

Sets the aglet property indicated by the key and the value.
Parameters:
key - the name of the aglet property.
value - the value to put
Throws: InvalidAgletException
if the aglet is not valid.

● **getPropertyKeys**

public final Enumeration getPropertyKeys() throws InvalidAgletException

Enumerates all the property keys.
Returns:
property key enumeration.
Throws: InvalidAgletException
if the aglet is not valid.

● **subscribeMessage**

public final void subscribeMessage(String name) throws InvalidAgletException

Subscribes to a named message.
Parameters:
name - the message kind.
Throws: InvalidAgletException
if the aglet is not valid.

● **unsubscribeMessage**

public final boolean unsubscribeMessage(String name) throws InvalidAgletException

Unsubscribes from a named message.
Parameters:
name - the message kind.
Returns:
true if the message kind was subscribed.
Throws: InvalidAgletException
if the aglet is not valid.

- **unsubscribeAllMessages**

public final void unsubscribeAllMessages() throws InvalidAgletException

Unsubscribes from all message kinds.
Throws: InvalidAgletException
if the aglet is not valid.

- **getImage**

public final Image getImage(URL url) throws IOException, InvalidAgletException

Gets an image
Throws: InvalidAgletException
if the aglet is not valid.

- **getImage**

public final Image getImage(URL url,
String name) throws IOException, InvalidAgletException

Gets an image
Throws: InvalidAgletException
if the aglet is not valid.

- **getAudioData**

public final AudioClip getAudioData(URL url) throws IOException, InvalidAgletException

Gets an audio data
Throws: InvalidAgletException
if the aglet is not valid.

- **getIconImage**

public Image getIconImage()

Returns an icon image that represents the aglet.

- **setProxy**

public final synchronized void setProxy(AgletProxy proxy)

Sets the proxy for the aglet. This cannot be set twice. Called by the system.
Parameters:
proxy - the proxy to set

- **setItinerary**

public final synchronized void setItinerary(Itinerary itinerary) throws AgletException

Assigns an Itinerary object to the aglet. This assignment is allowed only once during the life-time of the aglet.
Parameters:
itinerary - the Itinerary object.
Throws: AgletException
if the aglet was already assigned an itinerary.

Class aglet.AgletIdentifier

public final class AgletIdentifier
extends Object
implements Serializable

The AgletIdentifier class represents the uniq identifier given the aglet.

Constructors

AgletIdentifier

public AgletIdentifier(byte b[])

Constructs an aglet identifier with given byte array.

AgletIdentifier

public AgletIdentifier(String rep)

Methods

toByteArray

public byte[] toByteArray()

Returns byte array representation of the id. The copy of array is returned so that it cannot be altered.

toString

public String toString()

Returns a human readable form of the aglet identifier.
Returns:
the Aglet identity in text form.
Overrides:
toString in class Object

equals

public boolean equals(Object obj)

Compares two aglet identifiers.
Parameters:
obj - the Aglet to be compared with.
Returns:
true if and only if the two Aglets are identical.
Overrides:
equals in class Object

- **hashCode**

public int hashCode()

Returns an integer suitable for hash table indexing.
Returns:
hash table indexing integer.
Overrides:
hashCode in class Object

Class aglet.AgletProxy

public class AgletProxy
extends Object

Abstract class AgletProxy is a placeholder for aglets. The purpose of this class is to provide a mechanism to control and limit direct access to aglets.

Constructors

- **AgletProxy**

protected AgletProxy()

Methods

- **getAglet**

public abstract Aglet getAglet() throws InvalidAgletException

Gets the aglet that the proxy manages.
Returns:
the aglet
Throws: InvalidAgletException
if the aglet is not valid.
Throws: SecurityException
if you are not allowed to access the aglet.

- **getIdentifier**

public abstract AgletIdentifier getIdentifier() throws InvalidAgletException

Gets the aglet's identifier.
Returns:
the aglet's identifier.
Throws: InvalidAgletException
if the aglet is not valid.

● **getAgletClassName**

public abstract String getAgletClassName() throws InvalidAgletException

Gets the aglet's class name.
Returns:
the class name.
Throws: InvalidAgletException
if the aglet is not valid.

● **getCodeBase**

public abstract URL getCodeBase() throws InvalidAgletException

Gets the URL of the aglet's class.
Returns:
the class URL.
Throws: InvalidAgletException
if the aglet is not valid.

● **getProperty**

public abstract String getProperty(String key) throws InvalidAgletException

Gets the aglet property indicated by the key.
Parameters:
key - the name of the aglet property.
Returns:
the value of the specified key.
Throws: InvalidAgletException
if the aglet is not valid.

● **getProperty**

public abstract String getProperty(String key,
String defValue) throws InvalidAgletException

Gets the aglet property indicated by the key and default value.
Parameters:
key - the name of the aglet property.
defValue - the default value to use if this property is not set.
Returns:
the value of the specified key.
Throws: InvalidAgletException
if the aglet is not valid.

● **clone**

public abstract Object clone() throws CloneNotSupportedException

Clones the aglet and its proxy. Note that the cloned aglet will get activated. If you like to get cloned aglet which is not activated, throw ThreadDeath exception in the onClone method.
Returns:
the new aglet proxy what holds cloned aglet.
Throws: CloneNotSupportedException
if the cloning fails.
Throws: InvalidAgletException
if the aglet is invalid.
Overrides:
clone in class Object

● **dispatch**

public abstract AgletProxy dispatch(URL url) throws IOException, AgletException

● **dispose**

public abstract void dispose() throws InvalidAgletException

Disposes the aglet.
Throws: InvalidAgletException
if the aglet is invalid.

● **deactivate**

public abstract void deactivate(long milliseconds) throws InvalidAgletException

Deactivates the aglet. The system may store the aglet in the spool (disk or memory depending on the server). The aglet will be re-activated later (at the given time or manually).
Parameters:
milliseconds - duration of the aglet deactivating.
Throws: InvalidAgletException
if the aglet is not valid.
Throws: IllegalArgumentException
if the minutes parameter is negative.

● **sendMessage**

public abstract Object sendMessage(Message msg) throws InvalidAgletException, MessageNotHandledException, MessageException

● **sendAsyncMessage**

public abstract FutureReply sendAsyncMessage(Message msg) throws InvalidAgletException

● **isValid**

public abstract boolean isValid()

Checks if it's valid or not.
Returns:
true if the aglet is valid. false if not.

● **setAglet**

protected abstract void setAglet(Aglet aglet)

● **getText**

protected abstract String getText() throws InvalidAgletException

Gets the current content of the Aglet's message line.
Returns:
the message line.
Throws: InvalidAgletException
if the aglet is not valid.

● **getAgletContext**

protected abstract AgletContext getAgletContext() throws InvalidAgletException

● **getMessageManager**

protected abstract MessageManager getMessageManager() throws InvalidAgletException

Gets the aglet's message manager object.
Returns:
the method manager
Throws: InvalidAgletException
if the aglet is not valid.

● **setProperty**

protected abstract void setProperty(String key,
String value) throws InvalidAgletException

● **setItinerary**

protected abstract void setItinerary(Itinerary itinerary) throws InvalidAgletException

● **getItinerary**

protected abstract Itinerary getItinerary() throws InvalidAgletException

● **setText**

protected abstract void setText(String text) throws InvalidAgletException

Sets a aglet's text

- **subscribeMessage**

protected abstract void subscribeMessage(String name) throws InvalidAgletException

- **unsubscribeMessage**

protected abstract boolean unsubscribeMessage(String name) throws InvalidAgletException

- **unsubscribeAllMessages**

protected abstract void unsubscribeAllMessages() throws InvalidAgletException

- **getPropertyKeys**

protected abstract Enumeration getPropertyKeys() throws InvalidAgletException

Enumerates all the property keys.
Returns:
property key enumeration.
Throws: InvalidAgletException
if the aglet is not valid.

Class aglet.Itinerary

public class Itinerary
extends Object

The Itinerary is an abstract base class for control over an aglet's tour. Use this class to create specialized Itinerary classes. An Itinerary object is bound to an Aglet by the Aglet's setItinerary method. Any Itinerary object can be bound to at most one specific aglet (known as the traveller).

Variables

- **traveller**

protected transient AgletProxy traveller

- **FIRST**

public final static String FIRST

- **LAST**

public final static String LAST

- **NEXT**

  public final static String NEXT

- **PREVIOUS**

  public final static String PREVIOUS

Constructors

- **Itinerary**

  public Itinerary()

Methods

- **setTraveller**

  public final void setTraveller(AgletProxy proxy)

- **unsetTraveller**

  protected final void unsetTraveller()

- **initialize**

  protected void initialize()

- **go**

  public abstract void go(String symbolic_name) throws IOException, AgletException

- **isCurrent**

  public abstract boolean isCurrent(String symbolic_name) throws InvalidAgletException

- **isAvailable**

  public abstract boolean isAvailable(String symbolic_name) throws InvalidAgletException

- **reset**

  public abstract void reset() throws InvalidAgletException

- **keywords**

  public abstract String[] keywords()

Class aglet.Message

public class Message
extends Object
implements Serializable

The Message class is a object that holds its kind and arguments passed to the receiver. In handleMessage method on Aglet class, the reply of the request can be set if any.

Variables

- **arg**

  public Object arg

- **kind**

  public String kind

- **timestamp**

  public long timestamp

Constructors

- **Message**

  public Message(String kind)

  Constructs a message. The message object created by this constructor have a hashtable which can be used for argument-value pair.

  ```
  Message msg = new Message("stock-price");
  msg.setArg("company", "ibm");
  msg.setArg("currency", "dallar");
  Double d = (Double) proxy.sendMessage(msg);
  ```

  Parameters:
  kind - a kind of this message

- **Message**

  public Message(String kind,
  Object arg)

  Constructs a message with an argument value.
  Parameters:
  kind - a kind of this message
  arg - an argument of this message

Message

public Message(String kind,
int i)

Constructs a message with an argument value.
Parameters:
kind - a kind of this message

Message

public Message(String kind,
double d)

Constructs a message with an argument value.
Parameters:
kind - a kind of this message

Message

public Message(String kind,
float f)

Constructs a message with an argument value.
Parameters:
kind - a kind of this message

Message

public Message(String kind,
boolean b)

Constructs a message with an argument value.
Parameters:
kind - a kind of this message

Message

public Message(String kind,
char c)

Constructs a message with an argument value.
Parameters:
kind - a kind of this message

Message

public Message(String kind,
long l)

Constructs a message with an argument value.
Parameters:
kind - a kind of this message

Methods

- **setArg**

public void setArg(String name,
Object a)

Sets an argument value with associated name.
Parameters:
name - a name of this argument
a - a value of this argument

- **getArg**

public Object getArg()

- **getArg**

public Object getArg(String name)

Gets an argument value
Parameters:
name - a name of this argument
Returns:
a value of this argument

- **sendReply**

public void sendReply(Object arg)

Sets a reply to this message.

- **sendException**

public void sendException(Exception exp)

Sets a exception to this message.

- **sendReply**

public void sendReply()

Send a reply without sepcific value.

- **sendReply**

public void sendReply(int i)

- **sendReply**

public void sendReply(double d)

● **sendReply**

public void sendReply(float f)

● **sendReply**

public void sendReply(boolean b)

● **sendReply**

public void sendReply(char c)

● **sendReply**

public void sendReply(long l)

● **toString**

public String toString()

Overrides:
toString in class Object

Class ibm.aglets.patterns.Messenger

public final class Messenger
extends Aglet

Create a Messenger by calling the static method create. The messenger will get dispatched automatically. The messenger carries a message between two remote aglets. Upon reaching the host of the receiver aglet and sending the message, the messenger complete its job and so, it is disposed. If a Messenger cannot be dispatched, it is disposed.

Constructors

● **Messenger**

public Messenger()

Methods

● **create**

public static AgletProxy create(AgletContext context,
URL dest,
AgletIdentifier id,
Message message) throws IOException, AgletException

Creates a messenger.
Parameters:
context - the aglet context in which the messenger should be created.
dest - the host of the receiver aglet.
id - the receiver's agletIdentifier.
message - the message object.
Returns:
an aglet proxy for the messenger.
Throws: AgletException
if initialization fails.

● **create**

public static AgletProxy create(AgletContext context,
URL agletURL,
Message message) throws IOException, AgletException

Creates a messenger.
Parameters:
context - the aglet context in which the messenger should be created.
agletURL - the aglet URL of the receiver aglet.
message - the message object.
Returns:
an aglet proxy for the messenger.
Throws: AgletException
if initialization fails.

● **onCreation**

public synchronized void onCreation(Object object)

Initializes the messenger. The argument object containes the destination URL and the the message object.
Throws: AgletException
if initialization fails.
Overrides:
onCreation in class Aglet

● **run**

public void run()

Universal entry point for the messenger's execution thread.
Overrides:
run in class Aglet

Class ibm.aglets.patterns.Notifier

public class Notifier
extends Aglet

Create a notifier by calling the static method create. The notifier will get dispatched automatically. The notifier performs successive checks (at its destination) within a specified time duration. Upon every successfull check (one the encounters a change in a local state), it notifies its master. The notifier can be defined (see create) to complete its job after the first successive check (although its time duration has not been reached yet). If a notifier cannot be dispatched or it encounters an error during a check, it notifies its master and disposed itself.

Variables

- **NOTIFICATION**

  public final static int NOTIFICATION

- **EXPIRY**

  public final static int EXPIRY

- **EXCEPTION**

  public final static int EXCEPTION

- **MESSAGE**

  protected Object MESSAGE

  The protected variable that carries any messages that should go along with the notification back to the subscriber.

- **ARGUMENT**

  protected Object ARGUMENT

  The protected variable that carries any arguments for the checks that this notifier performs.

Constructors

- **Notifier**

  public Notifier()

Methods

- **initializeCheck**

  protected abstract void initializeCheck() throws Exception

This method should be overridden to specify any intialization before the checks performed by this notifier.
Throws: AgletException
if fails to complete.

- **doCheck**

protected abstract boolean doCheck() throws Exception

This method should be overridden to specify the check method for this notifier.
Returns:
boolean result of the check.
Throws: AgletException
if fails to complete.

- **getReceiver**

protected AgletIdentifier getReceiver()

Gets the URL of this notifier's receiver.

- **getOrigin**

protected URL getOrigin()

Gets the URL of the Origin of the aglet.

- **create**

public static AgletProxy create(URL url,
String source,
AgletContext context,
Aglet master,
Itinerary destination,
double interval,
double duration,
boolean stay,
Object argument) throws IOException, AgletException

Creates a notifier.
Parameters:
url - the URL of the aglet class.
source - the name of the aglet class.
context - the aglet context in which the notifier should be created.
master - the master aglet.
destination - the URL of the destination.
interval - the time in hours between to checks.
duration - the life time of the notifier.
stay - whether the notifier should remain after a notification.
argument - the argument object.

Returns:
an aglet proxy for the notifier.
Throws: AgletException
if the creation fails.

● **onCreation**

public synchronized void onCreation(Object object)

Initializes the notifier. Only called the very first time this notifier is created. The initialization argument includes the needed parameters for the checks as defined in create

Parameters:
obj - the initialization argument.
Throws: AgletException
if the initialization fails.
Overrides:
onCreation in class Aglet

● **run**

public void run()

Universal entry point for the notifier's execution thread.
Overrides:
run in class Aglet

Class ibm.aglets.patterns.Slave

public class Slave
extends Aglet

Create a slave by calling the static method create. The slave will get dispatched automatically.
Given an itinerary, the slave is travelled from one destination to another while performing local computation in every destination. The results of these local computations are accumulated and finally submitted to the slave's Master (the creator of the slave). During its tour the slave skips destinations which are not available (i.e. it cannot be dispatched to).
When a slave:
1) completes its tour.
2) encounters an error during a local computation.
3) cannot be dispatched to yet unvisited destinations.
it immediately *returns* to its origin host and submits the intermediate result.

Variables

- **RESULT**

protected Object RESULT

The protected variable that accumulates the results of the local tasks that this slave performs in every destination.

- **ARGUMENT**

protected Object ARGUMENT

The protected variable that carries an argument for the local task that the slave performs in every destination.

Constructors

- **Slave**

public Slave()

Methods

- **initializeJob**

protected abstract void initializeJob() throws Exception

This method should be overridden to specify initialization part for the job of the slave.
Throws: AgletException
if fails to complete.

- **doJob**

protected abstract void doJob() throws Exception

This method should be overridden to specify the local task of the slave.
Throws: AgletException
if fails to complete.

- **getMaster**

protected AgletIdentifier getMaster()

Gets the Aglet Idedntifier of the lave's master.

- **getOrigin**

protected URL getOrigin()

Gets the URL of the Origin of the aglet.

- **returnHome**

protected void returnHome() throws IOException, AgletException

Dispatch to the Origin

- **onReturn**

protected void onReturn()

Called when the slave returns to its origin.

- **create**

public static AgletProxy create(URL url,
String name,
AgletContext context,
Aglet master,
Itinerary itinerary,
Object argument) throws IOException, AgletException

Creates a slave.
Parameters:
url - the url of the aglet class.
name - the name of the aglet class.
context - the aglet context in which the slave should be created.
master - the master aglet.
itinerary - the itinerary object.
argument - the

argument

object.
Returns:
an aglet proxy for the slave.
Throws: AgletException
if initialization fails.

- **onCreation**

public synchronized void onCreation(Object object)

Initializes the slave. Only called the very first time this slave is created. The initialization argument includes four elements: the first is the master aglet, the second is the Slave's itinerary, the third is the initial value of the protected result variable and the fourth is the value of protected argument variable.

Parameters:
obj - the argument vector.
Throws: AgletException
if the initialization fails.
Overrides:
onCreation in class Aglet

- **setDefferedReturn**

public void setDefferedReturn(boolean b)

Define whether to defer the return of the slave to its origin.

- **run**

public void run()

Universal entry point for the slave's execution thread.
Overrides:
run in class Aglet

Class ibm.aglets.util.Arguments

public final class Arguments
extends Hashtable

The Argument class is a object that holds various kinds of objects as arguments.

Constructors

- **Arguments**

public Arguments()

Methods

- **putArgument**

public Object putArgument(String key,
Object value)

- **putArgument**

public Object putArgument(String kind,
int i)

- **putArgument**

```
public Object putArgument(String kind,
                          double d)
```

- **putArgument**

```
public Object putArgument(String kind,
                          float f)
```

- **putArgument**

```
public Object putArgument(String kind,
                          boolean b)
```

- **putArgument**

```
public Object putArgument(String kind,
                          char c)
```

- **putArgument**

```
public Object putArgument(String kind,
                          long l)
```

- **getArgument**

```
public Object getArgument(String key)
```

- **clone**

```
public Object clone()
```

Overrides:
clone in class Hashtable

Class ibm.aglets.util.SeqItinerary

public class SeqItinerary
extends Itinerary
implements Externalizable

The Itinerary class enables to define and take control over an aglet's tour.

Terminology:

tour : a set of destinations to visit.
place : a destination within a tour.
origin : the place which is considered as the original place from
which the tour is started.
handler: the place to which an aglet can be dispatched in case of
unexpected events. The origin is the default handler.
log : a record of messages to trace the aglet's tour.

Constructors

SeqItinerary

public SeqItinerary(URL destination)

Creates an Itineray with a single destination.
Parameters:
destination - the URL of the destinatibn to be visited.

SeqItinerary

public SeqItinerary()

SeqItinerary

public SeqItinerary(Vector destinations)

Creates an Itineray.
Parameters:
itinVector - a vector of URLs to visit.

Methods

writeExternal

public void writeExternal(ObjectOutput out) throws IOException

readExternal

public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException

setSkipUnavailableHost

public void setSkipUnavailableHost(boolean skip)

Configures so that it will skip the host which is not rearchable.
Parameters:
skip - true if you want to skip unavailable hosts. Default is false.

initialize

protected void initialize()

Bind an aglet (for the tour).
Parameters:
aglet - the aglet to be bound.
Overrides:

initialize in class Itinerary

- **gotoNextAvailable**

public void gotoNextAvailable() throws IOException, AgletException

Dispatch an aglet to the next AVAILABLE place in the tour.

- **gotoNext**

public void gotoNext() throws IOException, AgletException

Dispatch an aglet to the next place in the tour.

- **gotoByIndex**

public void gotoByIndex(int index) throws IOException, AgletException

Dispatch an aglet to a new place with a specific index.

- **reset**

public void reset()

Resets an aglet's tour as if the tour have not yet been started. The aglet is dispatched to the origin host.
Overrides:
reset in class Itinerary

- **gotoOrigin**

public void gotoOrigin() throws IOException, AgletException

Dispatch an aglet to the origin.

- **gotoErrorHandler**

public void gotoErrorHandler() throws IOException, AgletException

Dispatch an aglet to the handler.

- **logToString**

public String logToString()

Converts the log to a string.

- **clearLog**

public void clearLog()

Clear the log (remove all entries).

- **setHandler**

  public void setHandler(URL handler)

  Defines the URL of an host to which the aglet can be dispatched in cases of unexpected events during its tour.

- **setNumRetries**

  public void setNumRetries(int num)

  Defines maximum number of retries to dispatch an aglet to a new place.

- **setNumAllowedFailures**

  public void setNumAllowedFailures(int num)

  Defines maximum number of allowed failures to visit a place during a tour.

- **getCurrentURL**

  public URL getCurrentURL()

  Tells the URL of the current visited place on the tour.

- **isAtOrigin**

  public boolean isAtOrigin()

  Tells whether the aglet is currently located in the Origin host

- **isAtHandler**

  public boolean isAtHandler()

  Tells whether the aglet is currently visiting the Handler host.

- **isAtLastDestination**

  public boolean isAtLastDestination()

  Tells whether the aglet is currently visiting the last place in the tour. In case of an empty tour, it returns true.

- **getOrigin**

  public URL getOrigin()

  returns the URL of the Origin.

- **dispatchHandler**

protected void dispatchHandler(URL url) throws IOException, AgletException

Dispatch an aglet to a specific URL.
Throws: AgletException
if no traveller exists or dispatching is failed.

- **go**

public void go(String name) throws IOException, AgletException

"origin" and NEXT are supported.
Overrides:
go in class Itinerary

- **isCurrent**

public boolean isCurrent(String name)

"origin" and LAST are supported.
Overrides:
isCurrent in class Itinerary

- **isAvailable**

public boolean isAvailable(String name)

Currently, it always returns false
Overrides:
isAvailable in class Itinerary

- **keywords**

public String[] keywords()

Overrides:
keywords in class Itinerary

Apéndice B

API del Sistema

Las siguientes son las descripciones de las clases que forman parte del sistema.

Class unam.aas.aglet.AASAglet

public class AASAglet
extends StationaryAglet

Me encargo de administrar a todos los agentes móviles que realizan funciones específicas. Tengo mi propio gui y me encargo de desplegarlo cuando soy creado. En total manejo cuatro agentes móviles: Un agente que se encarga de monitorear los archivos que el administrador selecciona, en las máquinas que el también elige, un agente que actualiza archivos de configuración o directorios importantes, un agente que realiza un inventario actualizado de las máquinas que se administran, un agente que sirve como mensajero entre mis esclavos y yo. Heredo todas las características de un agente estacionario definidas en mi superclase StationaryAglet.

Constructors

- **AASAglet**

public AASAglet()

Methods

- **handleMessage**

public boolean handleMessage(Message msg)

Manejo los mensajes de los agentes móviles. Cuando un agente regresa me manda un mensaje para indicarme si tuvo éxito o tuvo algun problema.

Parameters:
msg - Mensaje entregado por el agente móvil
Returns:
boolean true si es un mensaje conocido por el aglet maestro
Overrides:
handleMessage in class Aglet

- **onCreation**

public void onCreation(Object o)

Método ejecutado cuando soy creado.

Parameters:
o - parámetro para inicializar al aglet
Overrides:
onCreation in class Aglet

- **message**

protected synchronized void message(Arguments message)

Método message para manejo de mensajes entregados por el aglet de monitoreo

Parameters:
message - argumento a través del cual se le notifica al aglet
maestro de las actividades realizadas por el aglet de monitoreo
Overrides:
message in class StationaryAglet

- **sendMAglet**

public void sendMAglet(Hashtable data)

Método para especificar la tarea a efectuar por el aglet de monitoreo

Parameters:
data - información requerida por el aglet de monitoreo

- **goMAglet**

protected void goMAglet(URL destination,
double interval,
double duration,
boolean stay,
Hashtable files,
int index)

Método para enviar al aglet de monitoreo a efectuar su tarea

Parameters:
destination - URL al que viajara el aglet
interval - define la frecuencia de monitoreo
duration - define el tiempo total de monitoreo
stay - define si el agente se mantiene o no notificando que un archivo sufrió cambios
files - nombres de los archivos a monitorear
index - índice del aglet de monitoreo

- **sendUAglet**

public void sendUAglet(Hashtable data)

Método para especificar la tarea a efectuar por el aglet de actualización

Parameters:
data - información requerida por el aglet de actualización

● **goUAglet**

protected void goUAglet(Vector itinerary,
Vector updatingFiles)

Método para enviar al aglet de actualización a efectuar su tarea

Parameters:
itinerary - vector donde se especifica el nombre de las máquinas que deberá visitar el aglet de actualización
updatingFiles - vector donde se especifican los nombres de los archivos a actualizar

● **sendSAglet**

public void sendSAglet(Hashtable data)

Método para especificar la tarea a efectuar por el aglet de inventario

Parameters:
data - información requerida por el aglet de inventario

● **goSAglet**

protected void goSAglet(Vector itinerary,
String path)

Método para enviar al aglet de inventario a efectuar su tarea

Parameters:
itinerary - vector donde se especifica el nombre de las máquinas que deberá visitar el aglet de actualización
path - nombre del archivo donde se encuentra la información correspondiente al inventario de la máquina

● **callback**

protected synchronized void callback(Object arg)

Método a través del cual el aglet esclavo le entrega los resultados al aglet maestro

Parameters:
arg - el resultado del trabajo realizado por el aglet esclavo
Overrides:
callback in class StationaryAglet

● **stopMAglet**

public void stopMAglet(Hashtable data)

Método para detener la ejecución de los agentes móviles de monitoreo

Parameters:
data - información requerida para detener al aglet de monitoreo

- **stopUAglet**

public void stopUAglet(Hashtable data)

Método para detener la ejecución del agente móvil de actualización

Parameters:
data - información requerida para detener al aglet de actualización

- **stopSAglet**

public void stopSAglet(Hashtable data)

Método para detener la ejecución del agente móvil de inventario

Parameters:
data - información requerida para detener al aglet de inventario

Class unam.aas.aglet.AASMonitoring

public class AASMonitoring
extends Notifier

AASMonitoring es un aglet móvil que se encarga de verificar si un archivo ha tenido alguna modificación. Si esto se cumple, el aglet crea un aglet Messenger y lo envía al aglet estacionario para reportarle dicho cambio. Cuando se vence el tiempo de monitoreo el aglet regresa y se lo notifica al aglet estacionario.

Constructors

- **AASMonitoring**

public AASMonitoring()

Methods

- **onReverting**

public void onReverting(URL url)

Este método es ejecutado cuando el agente es obligado a regresar

Parameters:
url - URL al que regresa el agente

Overrides:
onReverting in class Aglet

- **onArrival**

public void onArrival()

Este método es ejecutado cuando el agente es obligado a regresar

Parameters:
url - url al que regresa el agente
Overrides:
onArrival in class Aglet

- **initializeCheck**

protected void initializeCheck() throws AgletException

Este método especifica que es lo que debo de hacer al momento de llegar a la máquina remota. Esto lo realizo sólo una vez.

Throws: AgletException
Si fallo al tratar de realizar mi inicialización.
Overrides:
initializeCheck in class Notifier

- **doCheck**

protected boolean doCheck() throws AgletException

Este método especifica la tarea que debo de realizar de forma periódica.

Returns:
boolean El resultado de la revisión actual.
Throws: AgletException
Si fallo al tratar de hacer la revisión.
Overrides:
doCheck in class Notifier

Class unam.aas.aglet.AASStock

public class AASStock
extends Slave

AASStock es la clase del aglet móvil encargado de obtener la información correspondiente al inventario de cada una de las máquinas que se le especifiquen en su itinerario. El archivo de itinerario debe estar almacenado en la máquina remota.

Constructors

AASStock

public AASStock()

Methods

initializeJob

protected synchronized void initializeJob()

Este método inicializa mis variables para que comience mi trabajo una vez que he llegado a la máquina. Esto lo hago en cada máquina especificada en mi itinerario.

Overrides:
initializeJob in class Slave

doJob

protected synchronized void doJob() throws Exception

Este método especifica la tarea que debo realizar en la máquina remota una vez que se han inicializado mis variables. De ARGUMENT obtengo la ruta del archivo donde se encuentra la información relacionada con el inventario para convertirlo en arreglo de bytes y finalmente almacenarlos en un vector, el cual guarda los arreglos de bytes de los archivos de cada una de las máquinas registradas en mi itinerario.

Throws: Exception
Si no se me especifico el archivo o no existe
Overrides:
doJob in class Slave

onReverting

public void onReverting(URL url)

Este método es ejecutado cuando el agente es obligado a regresar

Parameters:
url - URL al que regresa el agente
Overrides:
onReverting in class Aglet

onArrival

public void onArrival()

Este método es ejecutado cuando el agente es obligado a regresar

Overrides:
onArrival in class Aglet

Class unam.aas.aglet.AASUpdating

public class AASUpdating
extends Slave

AASUpdating es la clase del aglet móvil encargado de actualizar archivos en cada una de las máquinas que se le especifique en su itinerario. Los archivos a actualizar deben estar almacenados en la máquina desde la cual se envía a este aglet.

Constructors

- **AASUpdating**

public AASUpdating()

Methods

- **initializeJob**

protected synchronized void initializeJob()

Este método inicializa mis variables para que comience mi trabajo una vez que he llegado a la máquina. Esto lo hago en cada máquina especificada en mi itinerario.

Overrides:
initializeJob in class Slave

- **doJob**

protected synchronized void doJob() throws Exception

Este método especifica la tarea que debo realizar en la máquina remota una vez que se han inicializado mis variables. De ARGUMENT obtengo los archivos en forma de arreglos de bytes que debo actualizar en cada una de las máquinas registradas en mi itinerario.

Throws: AgletException
Si no se me especifico el archivo
Overrides:
doJob in class Slave

- **onReverting**

public void onReverting(URL url)

Este método es ejecutado cuando el agente es obligado a regresar

Parameters:
url - URL al que regresa el agente
Overrides:
onReverting in class Aglet

● **onArrival**

public void onArrival()

Este método es ejecutado cuando el agente es obligado a regresar

Parameters:
url - url al que regresa el agente
Overrides:
onArrival in class Aglet

Class unam.aas.aglet.StationaryAglet

public class StationaryAglet
extends Aglet

Soy una clase abstracta que modela un agente estacionario maestro que coordina a todos los agentes móviles del sistema de administración. Me encargo de despachar y tomar decisiones dependiendo de los resultados obtenidos por los agentes esclavos.

Variables

● **windowAglet**

protected AASWindowAglet windowAglet

La ventana del agente estacionario.

Constructors

● **StationaryAglet**

public StationaryAglet()

Methods

● **onDispatching**

public synchronized void onDispatching(URL URL)

Método para evitar que alguien trate de desplazarme

Parameters:
URL - URL de la máquina a donde desean que me desplace

Overrides:
onDispatching in class Aglet

- **onReverting**

public void onReverting(URL URL)

Método para evitar que alguien trate de desplazarme

Parameters:
URL - URL de la máquina a donde desean que me desplace
Overrides:
onReverting in class Aglet

- **onDisposing**

public synchronized void onDisposing( )

Cuando soy destruido destruyo mi ventana

Overrides:
onDisposing in class Aglet

- **go**

protected void go(URL url)

Me encargo de iniciar el envío del agente móvil.

- **inError**

protected synchronized void inError(Object message)

Método para manejo de errores Debe ser sobrecargado por alguna de mis subclases para que procese adecuadamente los errores.

- **callback**

protected synchronized void callback(Object message)

Método para manejo de resultados. Debe ser sobrecargado por alguna de mis subclases para que procese adecuadamente los resultados

- **message**

protected synchronized void message(Arguments arguments)

Método message para manejo de mensajes. Debe ser sobrecargado

por alguna de mis subclases para que procese adecuadamente los mensajes.

Class unam.aas.gui.AASAboutDialog

public class AASAboutDialog
extends Dialog

Me encargo de desplegar la ventana de diálogo de información sobre la aplicación.

Constructors

- **AASAboutDialog**

```
public AASAboutDialog(Frame parent,
                      String titulo,
                      Image imagen)
```

Methods

- **insets**

```
public Insets insets()
```

Separación de los componentes del marco de la ventana

Returns:
Insets Devuelve una instancia de la clase Insets la cual determina la separación de los componentes con respecto al marco de la ventana
Overrides:
insets in class Container

- **action**

```
public boolean action(Event e,
                      Object arg)
```

Overrides:
action in class Component

- **gotFocus**

```
public boolean gotFocus(Event e,
                        Object arg)
```

Método para pasarle el focus al botón cuando la ventana tiene el focus

Overrides:
gotFocus in class Component

Class unam.aas.gui.AASCapturePanel

public class AASCapturePanel
extends Panel

Me encargo de crear un panel que contiene una lista de múltiple selección, un campo de texto, así como botones para agregar y borrar ítems dentro de la lista.

Constructors

- **AASCapturePanel**

public AASCapturePanel(String inspectionLabel)

Constructor Este constructor recibe una cadena como etiqueta. Esta etiqueta es desplegada dentro del panel.

Parameters:
inspectionLabel - etiqueta que indica si es un panel de captura de archivos o de hosts

Methods

- **getAddButton**

protected Button getAddButton()

- **getDelButton**

protected Button getDelButton()

- **getCaptureField**

protected TextField getCaptureField()

- **getInspectionList**

protected List getInspectionList()

Class unam.aas.gui.AASConfigurationDialog

public class AASConfigurationDialog
extends Dialog

Me encargo de desplegar la ventana de diálogo donde el usuario puede indicar los directorios donde desea guardar los archivos de log los reportes y los archivos de configuración. También puede escoger el visualizador de WEB que desee.

Methods

- **insets**

public Insets insets()

Overrides:
insets in class Container

- **action**

public boolean action(Event e,
Object arg)

Método para capturar el evento de aceptar y cancelar

Overrides:
action in class Component

- **gotFocus**

public boolean gotFocus(Event e,
Object arg)

Método para pasarle el focus al botón cuando la ventana tiene el focus

Overrides:
gotFocus in class Component

- **getLogDirectory**

public String getLogDirectory()

Método para devolver la ruta del logDirectory donde se guardan los archivos de log.

Returns:
String Es la ruta donde se almacenan los archivos de log.

- **getReportDirectory**

public String getReportDirectory()

Método que devuelve la ruta del reportDirectory donde se guardan los reportes.

Returns:
String. La ruta donde se almacenan los archivos de reporte.

- **getConfDirectory**

public String getConfDirectory()

Método que devuelve la ruta del confDirectory donde se guardan los archivos de configuración.

Returns:
String La ruta donde se almacenan los archivos de configuración.

Class unam.aas.gui.AASConsolePanel

public class AASConsolePanel
extends Panel3D

Extiendo las capacidades de Panel3D. Creo y arreglo componentes.

Constructors

AASConsolePanel

public AASConsolePanel(String names[],
Color colorBorde)

Me encargo de crear y colocar todos los componentes.

Parameters:
names - Este es un arreglo de cadenas. Cada cadena es el nombre de uno de los componentes. El orden es el siguientes:
names[0], nombre del botón de propiedades.
names[1], nombre del botón de start.
names[2], nombre del botón de stop.
names[3], nombre del botón de reportes.

Methods

setTextConsole

public void setTextConsole(String t)

Me encargo de escribir texto en la consola.

Parameters:
t - cadena que se desea desplegar en la consola

main

public static void main(String args[])

Tengo este método sólo con fines de prueba. De esta forma si se decide utilizar otro layoutManager para la distribución de los componentes que tengo solo es necesario utilizar este método para visualizarme.

Class unam.aas.gui.AASWindowAglet

public class AASWindowAglet
extends Frame

Me encargo de desplegar la interfaz gráfica del agente estacionario Soy responsable de desplegar el status de cada uno de los agentes y desplegar información con respecto al sistema.

Variables

MCONSOLE

public final static int MCONSOLE

UCONSOLE

public final static int UCONSOLE

SCONSOLE

public final static int SCONSOLE

Constructors

AASWindowAglet

public AASWindowAglet(String titulo,
Image image[],
AASAglet a)

Constructor Este constructor recibe un aglet como parámetro, así como el título de la ventana.

Parameters:
titulo - Título de la ventana
a - Aglet dueño de la ventana
image - Arreglo de imágenes a desplegar en los diferentes componentes.

Methods

main

public static void main(String args[])

Método que inicia la interfaz gráfica Sólo para fines de prueba. Se supone que el agente debe realizar esta tarea.

- **insets**

public Insets insets()

Separación de los componentes del marco de la ventana

Returns:
Insets Devuelve una instancia de la clase Insets la cual determina la separación de los componentes con respecto al marco de la ventana
Overrides:
insets in class Container

- **handleEvent**

public boolean handleEvent(Event event)

Capturo el evento de destruir la ventana

Parameters:
event - El evento que ha recibido la ventana
Returns:
boolean Devuelve true si procesa el evento false si el evento debe ser procesado por otro componente
Overrides:
handleEvent in class Component

- **action**

public boolean action(Event evt,
Object arg)

Capturo los eventos para desplegar la consola adecuada

Overrides:
action in class Component

- **setTextMessage**

public void setTextMessage(String message,
int consola)

Método para desplegar texto en las consolas y en tahiti.

- **command**

public void command(String argument,
int console)

Método para ejecutar el visualizador de WEB.

- **confirm**

  public void confirm()

  Método para confirmar la salida de la aplicación

- **quit**

  public void quit()

  Método para destruirme

- **writeToLog**

  public void writeToLog(String message)

  Método para escribir al archivo de log

- **saveConfiguration**

  public void saveConfiguration()

  Método para guardar la configuración del sistema

- **loadConfiguration**

  public void loadConfiguration()

  Método para cargar la configuración del sistema

Class unam.aas.gui.AASMonitoringDialog

public class AASMonitoringDialog
extends Dialog

Me encargo de desplegar la ventana de diálogo donde el usuario puede capturar los nombres de hosts y de archivos a monitorear, así como de permitir la configuración del agente para indicarle el itinerario que deberá cumplir

Variables

- **WSHOSTS**

  public final static String WSHOSTS

- **WSFILES**

  public final static String WSFILES

- **PCHOSTS**

  public final static String PCHOSTS

- **PCFILES**

  public final static String PCFILES

- **WSINTERVAL**

  public final static String WSINTERVAL

- **WSDURATION**

  public final static String WSDURATION

- **WSSTAY**

  public final static String WSSTAY

- **PCINTERVAL**

  public final static String PCINTERVAL

- **PCDURATION**

  public final static String PCDURATION

- **PCSTAY**

  public final static String PCSTAY

Methods

- **action**

  public boolean action(Event event,
  Object arg)

  Capturo los eventos para desplegar la consola adecuada, además de restablecer o cambiar la información de mis variables de instancia dependiendo de si el usuario presionó el botón de cancel o el de ok

  Parameters:
  event - acción que ocurre en el diálogo
  Overrides:
  action in class Component

- **insets**

  public Insets insets()

Separación de los componentes del marco de la ventana

Returns:
insets Devuelve una instancia de la clase Insets la cual determina la separación de los componentes con respecto al *marco* de la ventana
Overrides:
insets in class Container

● **getData**

public Hashtable *getData*()

Returns:
Hashtable Devuelvo los datos que almaceno

● **putData**

public void putData(String wsHosts[],
String pcHosts[],
String wsFiles[],
String pcFiles[])

Inicializo el contenido de las listas

Parameters:
wsHosts - arreglo de nombres de WSs a monitorerar
pcHosts - arreglo de nombres de PCs a monitorerar
wsFiles - arreglo de nombres de archivos de WS a monitorerar
pcFiles - arreglo de nombres de archivos de PC a monitorerar

● **getConfiguration**

public Hashtable getConfiguration()

Returns:
Hashtable devuelvo la configuración

Class unam.aas.gui.AASStockDialog

public class AASStockDialog
extends *Dialog*

Me encargo de desplegar la ventana de diálogo donde el usuario puede capturar los nombres de los hosts donde se buscaran los archivos que contienen la información de inventario del mismo, así como de permitir la configuración de la ruta donde se encuentran dichos archivos.

Variables

- **WSHOSTS**

public final static String WSHOSTS

- **PCHOSTS**

public final static String PCHOSTS

- **WSPATH**

public final static String WSPATH

- **PCPATH**

public final static String PCPATH

Methods

- **action**

public boolean action(Event event,
Object arg)

Capturo los eventos para desplegar la consola adecuada, además de restablecer o cambiar la información de mis variables de instancia dependiendo de si el usuario presionó el botón de cancel o el de ok.

Parameters:
event - acción llevada a cabo en el diálogo
Returns:
boolean true si manejo adecuadamente el evento, false si el evento debe ser manejado por la ventana a la que pertenezco.
Overrides:
action in class Component

- **insets**

public Insets insets()

Separación de los componentes del marco de la ventana

Returns:
Insets Devuelve una instancia de la clase Insets la cual determina la separación de los componentes con respecto al marco de la ventana
Overrides:
insets in class Container

- **getData**

public Hashtable getData()

Returns:
Hashtable devuelvo los datos que almaceno

- **putData**

public void putData(String wsHosts[],
String pcHosts[],
String wsPath,
String pcPath)

Inicializo el contenido de las listas y los campos

Parameters:
wsHosts - arreglo de nombres de WSs a monitorerar
pcHosts - arreglo de nombres de PCs a monitorerar
wsPath - nombre del archivo de WS a retraer
pcPath - nombre del archivo de PC a retraer

- **getConfiguration**

public Hashtable getConfiguration()

Parameters:
Hashtable - devuelvo la configuración

Class unam.aas.gui.AASUpdatingDialog

public class AASUpdatingDialog
extends Dialog

Me encargo de desplegar la ventana de diálogo donde el usuario puede capturar los nombres de hosts en donde se van a actualizar los archivos de configuración y los nombres de los archivos a actualizar.

Variables

- **WSHOSTS**

public final static String WSHOSTS

- **WSFILES**

public final static String WSFILES

- **PCHOSTS**

public final static String PCHOSTS

PCFILES

public final static String PCFILES

Methods

action

public boolean action(Event event,
Object arg)

Capturo los eventos para desplegar el panel adecuado

Parameters:
event - acción ocurrida en la ventana
Returns:
boolean true indica que si manejo el evento, false si el evento debe ser manejado por la ventana a la que pertenezco.
Overrides:
action in class Component

insets

public Insets insets()

Separación de los componentes del marco de la ventana

Returns:
Insets Devuelve una instancia de la clase Insets la cual determina la separación de los componentes con respecto al marco de la ventana
Overrides:
insets in class Container

getData

public Hashtable getData()

Returns:
Hashtable devuelvo los datos que almaceno

putData

public void putData(String wsHosts[],
String pcHosts[],
String wsFiles[],
String pcFiles[])

Inicializo el contenido de las listas

Parameters:
wsHosts - arreglo de nombres de WSs a monitorerar
pcHosts - arreglo de nombres de PCs a monitorerar

wsFiles - arreglo de nombres de archivos de WS a actualizar
pcFiles - arreglo de nombres de archivos de PC a actualizar

● **getConfiguration**

public Hashtable getConfiguration()

Returns:
Hashtable devuelvo la configuración

Class unam.aas.gui.AASShutdownDialog

public class AASShutdownDialog
extends Dialog

Me encargo de desplegar una ventana de diálogo para confirmar la terminación de mi ventana padre.

Constructors

● **AASShutdownDialog**

public AASShutdownDialog(Frame parent,
String titulo,
Image imagen)

Methods

● **insets**

public Insets insets()

Separación de los componentes del marco de la ventana

Returns:
Insets Devuelve una instancia de la clase Insets la cual determina la separación de los componentes con respecto al marco de la ventana
Overrides:
insets in class Container

● **action**

public boolean action(Event e,
Object arg)

Overrides:
action in class Component

● **gotFocus**

public boolean gotFocus(Event e,
Object arg)

Método para pasarle el focus al botón cuando la ventana tiene el focus

Overrides:
gotFocus in class Component

Class unam.aas.report.AASReporter

public class AASReporter
extends Object

Me encargo de generar los reportes a partir de un archivo de log provisto por el usuario.

Constructors

● **AASReporter**

public AASReporter(String inputFile,
String outputFile)

Constructor Este constructor recibe dos rutas de archivos

Parameters:
inputFile - archivo donde se encuentra la información a procesar
outputFile - archivo donde se guardara la información procesada

Methods

● **main**

public static void main(String astring[])

Método principal para fines de prueba

● **createMonitoringReport**

public void createMonitoringReport()

Método que procesa el archivo de log para generar el reporte correspondiente al monitoreo

● **createUpdatingReport**

public void createUpdatingReport()

Método que procesa el archivo de log para generar el reporte

correspondiente a la actualización

- **createStockReport**

public void createStockReport()

Método que procesa el archivo de log para generar el reporte correspondiente a el inventario

- **readMonitoring**

public void readMonitoring(DataInputStream dis)

Método que lee el archivo de log para procesar la información correspondiente al monitoreo

Parameters:
dis - Flujo de datos del archivo de log

- **writeMonitoring**

public void writeMonitoring(DataOutputStream dos)

Método que guarda los resultados en el archivo Monitoring.html

Parameters:
dos - Flujo donde se guarda la información procesada

- **readUpdating**

public void readUpdating(DataInputStream dis)

Método que lee el archivo de log para procesar la información correspondiente a la actualización

Parameters:
dis - Flujo de datos del archivo de log

- **writeUpdating**

public void writeUpdating(DataOutputStream dos)

Método que guarda los resultados en el archivo Updating.html

Parameters:
dos - Flujo donde se guarda la información procesada

- **readStock**

public void readStock(DataInputStream dis)

Método que lee el archivo de log para procesar la información correspondiente al inventario

Parameters:
dis - Flujo de datos del archivo de log

- **writeStock**

public void writeStock(DataOutputStream dos)

Método que guarda los resultados en el archivo Stock.html

Parameters:
dos - Flujo donde se guarda la información procesada

Class unam.aas.report.HTMLGenerator

public class HTMLGenerator
extends Object

Me encargo de dar formato en HTML al texto que me proporcionan

Constructors

- **HTMLGenerator**

public HTMLGenerator()

Constructor Como constructor me encargo de proporcionar los ítems de HTML necesarios para dar formato al texto.

Methods

- **main**

public static void main(String astring[])

Método principal para fines de prueba

- **putHeader**

public String putHeader()

Me encargo de poner el encabezado de HTML al reporte.

Returns:
String devuelvo la cadena del encabezado.

- **putEnd**

public String putEnd()

Pongo las etiquetas necesarias para indicar el final de documento HTML.

Returns:
String devuelvo la cadena del fin de documento

- **putReferences**

public String putReferences(String task)

Coloco las referencias hacia los reportes de las tareas a las cuales no representa mi documento.

Parameters:
task - indica la tarea a la cual representa mi reporte
Returns:
String devuelvo la cadena de las referencias

- **putTitle1**

public String putTitle1(String s)

Parameters:
s - la cadena a la cual se le dará formato.
Returns:
String devuelvo la cadena con el formato: H1,color #0000FF.

- **putTitle2**

public String putTitle2(String s)

Parameters:
s - la cadena a la cual se le dará formato.
Returns:
String devuelvo la cadena con el formato: H2,B,I,color #FF0000,size 3.

- **putList**

public String putList(String s)

Parameters:
s - la cadena a la cual se le dará formato.
Returns:
String devuelvo la cadena dentro de una lista: UL.

- **putList1**

public String putList1(String s)

Parameters:
s - la cadena a la cual se le dará formato.
Returns:
String devuelvo la cadena con el formato: B,size 2.

- **putList2**

public String putList2(String s)

Parameters:
s - la cadena a la cual se le dará formato.
Returns:
String devuelvo la cadena dentro de una lista: LI con formato: color #000080.

- **putItem**

public String putItem(String s)

Parameters:
s - la cadena a la cual se le dará formato.
Returns:
String devuelvo la cadena dentro de una lista: LI con formato: B, size 2.

- **putLine**

public String putLine()

Returns:
String devuelvo la cadena que genera una linea: HR con formato: size 4,width 100%.

Class unam.aas.util.Configuration

public class Configuration
extends Object

Me encargo de generar los reportes a partir de un archivo de log provisto por el usuario.

Constructors

- **Configuration**

public Configuration(String outputFile,
Hashtable Mconf,
Hashtable Uconf,
Hashtable Sconf)

Constructor Este constructor recibe dos rutas de archivos

Parameters:
inputFile - archivo donde se encuentra la información a procesar

- **Configuration**

public Configuration(String inputFile)

Constructor Este constructor recibe una ruta de archivo

Parameters:
inputFile - archivo donde se encuentra la información de configuración del sistema

Methods

- **main**

public static void main(String astring[])

Método principal para fines de prueba

- **saveConfiguration**

public void saveConfiguration()

Método para guardar la configuración del sistema

- **loadConfiguration**

public Hashtable loadConfiguration()

Método para cargar la configuración del sistema

- **loadMonitoringConfiguration**

public Hashtable loadMonitoringConfiguration()

Método para cargar la configuración del sistema

- **loadUpdatingConfiguration**

public Hashtable loadUpdatingConfiguration()

Método para cargar la configuración del sistema

- **loadStockConfiguration**

public Hashtable loadStockConfiguration()

Método para cargar la configuración del sistema

Class unam.aas.util.ImagePanel

public class ImagePanel
extends Panel

Me encargo de desplegar una imagen dentro del panel. El usuario puede indicarme que imagen desea desplegar

Constructors

ImagePanel

public ImagePanel(Image i)

Constructor Este constructor se encarga de cargar las imágenes que serán desplegadas en el panel e inicializa la variable image.

Parameters:
path - Ruta que indica la ubicación de la imagen (gif o jpeg)

Methods

paint

public void paint(Graphics g)

Sobrecargo el método paint para desplegar la imagen.

Overrides:
paint in class Container

main

public static void main(String args[])

Tengo este método sólo con fines de prueba. De esta forma si se desea hacerme modificaciones solo es necesario utilizar este método para visualizarme.

Class unam.aas.util.MultipleImagePanel

public class MultipleImagePanel
extends Panel

Me encargo de desplegar una imagen dentro del panel. El usuario puede indicarme que imagen desea desplegar

Variables

IMAGEBLACK

public final static int IMAGEBLACK

IMAGEBLUE

public final static int IMAGEBLUE

IMAGERED

public final static int IMAGERED

Constructors

MultipleImagePanel

public MultipleImagePanel(Image imageA[])

Constructor Este constructor se encarga de cargar las imágenes que serán desplegadas en el panel e inicializa la variable image.

Methods

paint

public void paint(Graphics g)

Sobrecargo el método paint para desplegar la imagen.

Overrides:
paint in class Container

changeImage

public void changeImage(int imageIndex)

Cambio la imagen que tengo que desplegar. Y me encargo de repintarme.

Parameters:
color - Me indica qué imagen es la que se desea desplegar Es de tipo int. Colores permitidos:
AboutMultipleImagePanel.IMAGEBLACK
AboutMultipleImagePanel.IMAGEBLUE
AboutMultipleImagePanel.IMAGERED

main

public static void main(String args[])

Tengo esto método solo con fines de prueba. De esta forma si se desea hacerme modificaciones solo es necesario utilizar este método para visualizarme.

Class unam.aas.util.Panel3D

public class Panel3D
extends Panel

Me encargo de crear un panel que añade un marco en 3D. Mi constructor necesita un color para pintar el marco.

Variables

- **color**

protected Color color

Variable donde almaceno el color del marco

Constructors

- **Panel3D**

public Panel3D(Color colorBorde)

Methods

- **paint**

public void paint(Graphics g)

Me encargo de desplegar mi marco de un cierto color.

Parameters:
g - este es el contexto gráfico que me pertenece este parámetro es proporcionado por el ambiente cuando la ventana es desplegada. No se requiere invocar explícitamente.
Overrides:
paint in class Container

- **insets**

public Insets insets()

Defino la separación de los componentes de mis bordes.

Overrides:
insets in class Container

Glosario

A

alfa, versión
Es una liberación de software, la cual puede tener cambios muy radicales entre una liberación y otra. Generalmente las versiones alfa son inestables

Aglets
Son agentes desarrollados por IBM que contemplan las características de movilidad, autonomía y comunicación.

ANS (Agent Name Service)
Es un paquete que contiene las clases necesarias para crear un servidor de nombres de agentes dentro del framework del Java Agent Template. Por medio de este servidor se puede saber qué agentes existen en la red y la máquina donde se encuentran actualmente ejecutando.

AOP (Agent Oriented Programming)
Es un paradigma que considera el concepto de agentes como parte esencial de la programación, para apoyar este tipo de programación se han desarrollado algunos lenguajes especializados tales como telescript y Safe-Tcl-Tk.

API (Aplication Programming Interface)
Es un conjunto de especificaciones que indican la forma exacta en que se debe y puede interactuar o usar un sistema de software.

applet
Es un programa que debe ser ejecutado por medio de un visualizador de WWW.

appletviewer
Es un programa que permite visualizar applets. Este programa es proporcionado con el JDK.

ARPA KSE:
Grupo de personas dedicadas a la investigación en el área de agentes.

ATP (Agent Transfer Protocol)
Es un protocolo a nivel de aplicación independiente de lenguaje y plataforma que define la forma en que deben ser transferidos los agentes a través de una red.

AWT (Abstract Windowing Toolkit)
Es un paquete de clases que implementa los componentes de interfaz de usuario más comunes, tales como ventanas, botones, menús, y otros.

B

beta, versión
Es una liberación de software, cuyos cambios no agregan funcionalidad al sistema sino más bien tratan de corregir posibles errores que pueda presentar.

bootstrap
Secuencia de instrucciones a ejecutarse durante la fase de inicio en ciertos sistemas de cómputo.

BSD (Berkeley Software Distribution)
Siglas asociadas al software proporcionado por la Universidad de Berkeley en California.

ByteCodes
Es un conjunto de instrucciones similares al código ensamblador de algunas máquinas pero que no es específico de algún procesador.

C

C++
Es un lenguaje de programación orientado a objetos híbrido, basado en el lenguaje C.

callbacks
Son funciones o métodos ejecutados en respuesta a un cierto evento o solicitud, por parte de un agente externo al sistema.

clase
Es un objeto que define los métodos y atributos para un tipo particular de objeto. Todos los objetos de una clase dada son idénticos en forma y comportamiento pero contienen diferentes datos en sus variables.

clipboard
Es una área de memoria especial reservada para compartir información entre procesos. En esta área todos los procesos pueden escribir o leer.

CORBA (Common Object Request Broker Architecture)
Es una arquitectura de software que considera el intercambio de objetos hechos en diferentes lenguajes entre diversos sistemas.

criptografía
Es un área de las matemáticas dedicada al estudio de algoritmos para el cifrado de datos.

D

daemons
Programas que son ejecutados en background y que por lo regular prestan algún servicio a otros procesos.

DAI
Acrónimo para Distributed Artificial Intelligence

DB2
Manejador de bases de datos relacionales desarrollado por IBM.

DPS
Acrónimo para Distributed Problem Solutions

drag and drop
Es la facilidad que ofrecen algunos sistemas de software de "tomar", "arrastrar" y "soltar" algun componente dentro de su interfaz gráfica.

E

encapsulamiento
Es la técnica que permite que los detalles internos de un módulo no sean accesibles por otros módulos, protegiéndolo de interferencias exteriores y protegiendo a los otros módulos de confiar en detalles de implementación que pueden cambiar con el tiempo.

F

FTP (File Transfer Protocol)
Es un protocolo que permite la transferencia de archivos entre distintos sistemas de cómputo que utilizan TCP/IP como protocolo de transporte.

Framework
Es un conjunto de bibliotecas especializadas que sirven de apoyo para la creación de nuevos sistemas.

G

Garbage collection
Es un mecanismo que permite la liberación de memoria de manera automática sin la intervención directa del programador.

gateways
Son dispositivos de red que permiten la transferencia de información entre dos o más redes con igual o distinto protocolo de transporte..

GUI
Acrónimo para Graphic User Interface

H

herencia
Un mecanismo mediante el cual las clases pueden hacer uso de los métodos y variables definidas en todas las clases por encima de élla siguiendo su ruta dentro de la jerarquía de clase.

html (Hipertext Markup Language)
Es un lenguaje que permite la creación de documentos los cuales pueden tener referencias (hipertexto) hacia otros recursos (documentos, imágenes, programas, etc.) locales o remotos.

HTTP (Hipertext Transfer Protocol)
Es un protocolo a nivel de aplicación que permite la tranferencia de documentos HTML u otros recursos referenciados por dichos documentos.

I

Instancia
Un término usado para referirse a un objeto que pertenece a una clase en particular.

Intranets
Un concepto que hace referencia a la creación de una red a nivel corporativo que ofrece los servicios encontrados en Internet, con el objetivo de mejorar la funcionalidad de la empresa.

J

J-AAPI
acrónimo para Java Agent Aplication Program Interface

JAT (Java Agent Template)
Es un conjunto de clases que permite la creación de agentes con cierta capacidad de "aprendizaje".

Java
Lenguaje orientado a objetos desarrollado por Sun Microsystems y que ha sido utilizado principalmente para la creación de applets.

javac
Compilador de java. Este programa se encarga de generar el bytecode que posteriormente será interpretado por la máquina virtual de java para ejecutar el programa.

javadoc
Programa que permite la generación de documentación (API) en formato HTML del código de la aplicación de forma automática.

javah
programa que permite la creación de métodos nativos que serán invocados desde java.

JavaOS
Es un sistema operativo compacto que ejecuta aplicaciones java sobre dispositivos tales como computadoras en red o teléfonos celulares.

JAR
Es un formato de archivo independiente de plataforma que contiene información de varios archivos dentro de uno solo.

JDBC (Java Database Conectivity)
Es un API que define la forma en que las aplicaciones java se deben comunicar con controladores o manejadores de distintas bases de datos.

JDK (Java Developer's Kit)
Es un kit de desarrollo que incluye herrramientas para desarrollar aplicaciones y applets hechos en el lenguaje java.

JoDax
Es un conjunto de clases que permite a los agentes creados con el Aglets WorkBench tener acceso a bases de datos.

JOLT
Es un proyecto que tiene por objetivo la implementación de versiones de dominio público tanto del intérprete como del compilador de java.

JIT (just-in-time compilation)
Es una tecnología que acelera la ejecución de programas basados en el concepto de bytecode.

K

KQML (Knowledge Query and Manipulation Language)
Es un conjunto de protocolos y un lenguaje de alto nivel que permiten el intercambio de información entre sistemas cooperativos.

L

LAN
Acrónimo para Local Area Network (red de área local)

Lisp
Lenguaje de programación basado en inferencias ampliamente utilizado en el campo de la inteligencia artificial.

Lycos
Es un sistema que permite la búsqueda de información dentro de Internet.

M

makefile
Archivo utilizado por el comando make de unix que especifica relaciones entre archivos (principalmente archivos fuentes) y la manera de generar otros a partir de éstos.

maquina virtual
ver VM.

marshal
Es un técnica que permite la codificación de objetos activos en memoria con el fin de transferirlos por la red o guardarlos en algún otro medio.

microjava
Especificación de la primera generación de microprocesadores dedicados a la ejecución de aplicaciones hechas en java.

MIT
Acrónimo para Massachusets Institute of Technology

multicast
Es una técnica que permite enviar de forma asíncrona un mensaje a un grupo específico de agentes.

Multithread
Múltiples hilos de control. Es una técnica que permite la ejecución concurrente de varios hilos de control dentro de un proceso.

O

objeto
Un paquete de software que contiene una colección de datos relacionados (en forma de variables) y métodos (procedimientos) para manipular esos datos.

Object-REXX
Una extensión del lenguaje REXX desarrollado por IBM para que soporte el concepto de objetos.

ODBC (Open Database Conectivity)
Es una interfaz para accesar datos en un ambiente heterogéneo de sistemas manejadores de bases de datos relacionales y no relacionales. Éste proporciona una forma independiente de proveedor para accesar los datos.

OMG (Object Managment Group)
Un grupo de industrias dedicado a promover la tecnología orientada a objetos y conseguir la estandarización de ésta.

ontology
Un sistema de conceptos/vocabulario usados como primitivas para la construcción de sistemas de inteligencia artificial.

OSI (Open Systems Interconnection)
Un conjunto de estándares que definen reglas a las cuales se deben de apegar diversos servicios en diferentes niveles de una red .

P

patterns (patrones)
Un patrón describe una situación en la cual diversas clases cooperan en una cierta tarea y definen una organización específica y de comunicación. La descripción de un patrón debe contener no solamanete la información estructural (clases, sus relaciones, y comunicación), sino también el propósito del patrón, los problemas que resuelve, y las condiciones necesarias para su implementación exitosa.

PDA (Personal Digital Assistance)
Es un sistema de información personal que sirve de apoyo para la administración y realización de las labores diarias.

perl
Es un lenguaje interpretado optimizado para el procesamiento de archivos de texto, extracción de información de estos archivos e impresión de reportes basados en la información. También es un buen lenguaje para muchas tareas de administración de sistemas.

polimorfismo
La habilidad de ocultar diferentes implementaciones detrás de una interfaz común, simplificando la comunicación entre los objetos.

proxy
Es la autoridad de algo o alguien de actuar en representación de otro.

Python
Lenguaje de programación interpretado basado en scripts, diseñado para poder fácilmente utilizar programas hechos con otros lenguajes de programación.

R

REXX
Lenguaje de programación desarrollado por IBM para las plataformas OS/2 y AIX.

RMI (Remote Method Invocation)
Tecnología que permite la invocación de métodos de objetos que se encuentran ejecutando en diferentes máquinas virtuales.

RP
acrónimo para Remote Programming (programación remota)

RPC
acrónimo para Remote Procedure Call (Llamada a procedimiento remoto).

S

safe-tcl-tk
Es una extensión al lenguaje Tcl-Tk, la cual deshabilita las funciones de Tcl que podrían causar daño en la máquina donde se ejecuta un script obtenido de la red y el cual no es confiable.

script
Serie de comandos que son interpretados.

Serialización de objetos
Consiste en la codificación de objetos para convertirlos en un flujo de bytes, el cual puede ser utilizado para mandar el objeto por la red o a disco.

ServerSocket
Es un socket que se encarga de monitorear la red por posibles solicitudes de conexión provenientes de otra máquina.

shell
Un intérprete de comandos y lenguaje de programación en sistemas Unix.

SMTP (Simple Mail Transfer Protocol)
Es un protocolo que especifica las funciones básicas de transferencia de correo entre computadoras basadas en IP. SMTP utiliza TCP para establecer las conexiones.

sockets
Es la unión de la dirección IP de una máquina dada y un puerto de servicio asociado a una aplicación especifica. Un socket sirve como un medio de comunicación entre dos procesos.

softbot
Un agente que interactúa con un ambiente de software al usar comandos e interpretar la información proveniente de este ambiente.

Sparc
Arquitectura de microprocesadores basada en RISC desarrollada por SUN Microsystems.

Spiders
Es un programa que explora de forma autónoma la estructura del WWW y toma alguna acción con base en la información encontrada. Esta acción puede ser tan simple como contar el número de objetos encontrados, o tan compleja como indexar el contenido de un texto completo.

SQL (Structured Query Language)
Lenguaje estándar para la creación de consultas en una base de datos relacional.

stand-alone
Este término se aplica a toda aplicación que no requiere de ninguna otra para su ejecución.

T

Tahiti
Es un programa de aplicación que se ejecuta como un servidor de agentes y proporciona una interfaz de usuario para monitoreo, creación, envío y eliminación de agentes.

Tazza
Administrador gráfico de aglets que permite la creación de aglets por medio de componentes y de forma visual.

TCP/IP (Transfer Control Protocol/ Internet Protocol)
Conjunto de protocolos a nivel de transporte y red respectivamente que son utilizados normalmente en sistemas Unix.

telescript
Lenguaje de programación orientado a objetos específico para la creación de agentes.

Template
Es un esqueleto que define la estructura básica que debe de cumplir cualquier implementación basada en éste.

thread
hilo de control que define el flujo de ejecución dentro de un proceso.

U

UNICODE
Es un sistema de codificación de caracteres de 16 bits diseñado para soportar el intercambio, procesamiento y despliegue de los textos escritos en diversos lenguajes del mundo moderno.

URL (Uniformal Resource Locator)
Es un formato estándar utilizado en Internet para la ubicación de algún recurso en la red. El formato es: protocolo://recurso:puerto

USENIX SAGE
Organismo a nivel mundial que trata asuntos relativos a la administración de sistemas Unix.

V

visualizador
Herramienta de software que permite desplegar documentos en formato HTML.

VM (Virtual Machine)
Es un intérprete que ejecuta el bytecode generado previamente por un compilador.

W

WAN
Acrónimo para Wide Area Network (red de área amplia)

WebCrawlers
Son programas que pueden ser usados para buscar información sobre el World Wide Web.

WWW (World Wide Web)
Servicio de información a nivel mundial que se basa en la utilización de hipertexto.

X

XDR (Exchange Data Representation)
Es un estándar para la descripción y codificación de datos. Es útil para transferir datos entre diversas arquitecturas de cómputo. Usa un lenguaje para describir formato de datos. Este lenguaje solamente puede ser usado para describir datos, ya que no es un lenguaje de programación

Referencias

Direcciones URL

JavaSoft Home Page
http://www.javasoft.com/

Java(TM) Distributed Systems
http://chatsubo.javasoft.com/current/index.html

Java-To-Go
http://ptolemy.eecs berkeley.edu/dgm/javatools/Java-to-go/

UMBC AgentNews
http://www.cs.umbc.edu/agents/

IBM Aglets Workbench - Home Page
http://www.trl.ibm.co.jp/aglets/

Aglets Workbench list
http://www.javalounge.com/

Documentos

Intelligent Agents: Theory and Practice
Michael Wooldridge
Department of Computing
Nicholas R. Jennings
Department of Electronic Engineering
Manchester Metropolitan University, United Kingdom

An infrastructure for mobile agents: requirements and architecture
Anselm Lingnau
Oswald Drobnik
Frankfurt am Main Germany

Active Defense of a Computer System using Autonomous Agents
Marl Crosbie
Gene Spafford
Dept. of Computer Science
Purdue University

Is it an Agent, or just a Program?:
A taxonomy for Autonomous Agents
Stan Franklin and Art Graesser

Institute for Intelligent Systems
University of Memphis

An Architecture for Information Agents
Donald P McKay Pastor and Robin McEntire
Loral Defense Systems
Tim Finin
Computer Science and Electrical Engineering
University of Maryland, Baltimore County

Mobile Agent Security and Telescript
Joseph Tardo and Luis Valente
General Magic, Inc.

Agent, Services, and Electronic Markets:
How do they Integrate?
M. Merz, W.Lameradorf
University of Hamburgo, Department of Computer Science
Hamburgo, Germany

Secret Agents-
A Security Architecture for the KQML Agent Communication Language
Chelliah Thirunavukkarasu
Enterprise Integration Technologies, California
Tim Finin and James Mayfield
Computer Science and Electrical Engineering
University of Maryland Baltimore County