Apéndice C.
Más sobre Aglets

IBM de Japón ha desarrollado el concepto de *aglet* basado en Java. El aglet extiende el modelo de código móvil que presentan los applets de Java. Así como sucede en un apples, los archivos .class de los aglets pueden migrar a través de las redes, pero cuando los aglets migran también lo trace su estado. Un apples es código que puede moverse a través de las redes desde un servidor hacia el cliente. Un aglet es un programa Java (código y estado) que puede moverse desde una máquina anfitriona hacia otra, pero no solo eso, sino que pueden visitar varias máquinas anfitrionas secuencialmente o al azar y regresar luego a la máquina desde la cual migró inicialmente si así es diseño.[Castellanos y Sandoval 1997].

Un aglet para su ejecución requiere una aplicación anfitriona Java o servidor de agentes, ejecutándose en cada computadora antes de que el aglet pueda visitar la máquina. Cuando los aglets viajan a través de la red, migran desde un anfitrión de aglets a otro. Cada anfitrión ejecuta un administrador de seguridad pare reforzar restricciones sobre las actividades de aglets riesgosos. Los anfitriones cargan aglets a través de cargadores tipo class que saben como recuperar los archivos .class así como el estado desde un anfitrión remoto.

El Aglets Workbench (AWB) es el conjunto de clases y herramientas necesarias para la construcción de agentes móviles. E1 AWB esta integrado por:

1. Aglet FrameWork.
2. Agent Transfer Protocol.
3. Tazza.
4. Tahiti.
5. Fiji.

Aglet Framework. El componente principal del AWB, es el Aglet Framework, esté esta basado en el lenguaje de programación Java. Este ambiente introduce el concepto de aglet (agent applet), un agente móvil escrito en Java que es una clase abstracta de agente móvil.

El Aglet Framework provee un mecanismo de seguridad extensible. El lenguaje Java provee la primera capa de seguridad ya que el verificador

bytecode de Java checa el formato de código y generá chequeos de consistencia sobre este código. La siguiente capa de protección es el administrador de seguridad (Security Manager) que permite a los mismos programadores establecer sus propios mecanismos de protección. La última capa de protección es el mismo API de seguridad de Java, esta interfaz de programación permite criptografía, firmas digitales, encriptación y autentificación [Lange y Chang 1996].

Agent Transfer Protocol. Para hacer posible la transferencia de agentes, los *aglets* utilizan un prótocolo de transferencia de agentes, el ATP y actualmente esta disponible la primera versión de este llamado ATP/O. 1. Este prótocolo esta basado en los URL´s, y se enfoca principalmente a Internet y otrece servicios independientes de la plataforma y de lenguaje. ATP posee clases altamente portables que proveen un API estándar para crear servidores ATP, para conectar sitios ATP y generar respuestas a requerimientos ATP.

Tazza. En la actualidad existen varios compiladores que integran todo un ambiente de desarrollo visual como Visual Basic, PowerBuilder y Visual C++. Este ambiente de desarrollo en el AWB se llama Tazza. Este componente aun no se ha liberado en las versiones del AWB y se promete que estará disponible en versiones subsecuentes [Lange y Chang 1996].

Tahiti. El AWB contiene un administrador visual de agentes y trabaja conjuntamente con el aglet framework, este componente es el anfitrión del aglet. Este administrador es una interfaz gráfica hecha en Java que permite monitorear y controlar la ejecución de aglets. Es el componente esencial para poder utilizar los aglets, ya que abre un puerto de comunicaciones, como el que utilizan los demonios HTTP que por lo común es el puerto 80. Esto es necesario para que el ATP pueda contactar otros sitios que funcionan como este prótocolo [Lange y Chang 1996].

Fiji. Fiji es capaz de crear applets (Fiji applets) los cuales corren en contextos de aglets y puede crear, despachar o retractar aglets desde páginas Web. Fiji se compone de una biblioteca de aglets Fiji que es una biblioteca de clases AWB extendidas al visualizador Web, similar a un plugin y de un Kit Fiji que es una biblioteca class y un módulo de ruteo. ambos deben ser instalados en el servidor de Web.

El J-ATCI IBM introdujo toda una tecnología para hacer posible el envío de agentes móviles a través de las redes de una manera estándar. El J-ATCI es una de las partes fundamentales del AWB.

El J-ATCt (Java Aglet Transfer and Communication Interface) es un estándar independiente del protocolo pare comunicar o mover agentes dentro de una red. La implementación de J-ATCI es totalmente ajena a la interfaz, esta consiste de dos paquetes Java: el paquete atci con sus clases e interfaces y una implementación para una red en especial xxx.atci. IBM tiene el ibm.atci que soporta los protocolos ATP y HTTP, y estas son sus

ventajas:

Simplicidad: El J-ATCI es fácil de usar.

Extensibilidad: El J-ATCI es abierto a cualquier prótocolo de comunicación.

Independencia del sistema: Los agentes implementados en J-ATCI son capaces de ser transferidos a otros nodos y de comunicarse entre ellos.

Conceptos

Un *contexto* es el lugar de trabajo de un aglet. Es un objeto estacionario que provee los medios pare la administración de los aglets en un ambiente de ejecución uniforme en donde el sistema anfitrión esta seguro en contra de agentes maliciosos. Un nodo en una red puede tener múltiples contextos.

Un *proxy* es el representante de un aglet. Este sirve como un escudo para el aglet al cual protege del acceso directo hacia sus nodos públicos. El objeto proxy provee transparencia en la localidad del aglet.

Un mensaje es un objeto que es intercambiado entre aglets. Este puede ser pasado como parámetro de una forma sincrona o asincrona entre aglets, por tanto puede ser usado por los aglets pare la colaboración e intercambio de información.

Un *administrador de mensajes* permite el control de concurrencia en la administración de mensajes por parte de un aglet.

Un *itinerario* es el plan de viaje de un aglet.

Un *identificador* es un objeto atado a cada aglet. Este identificador es globalmente único e inmutable a lo largo de la vida del aglet.

El J-AAPI. El Java Aglet Application Programming Interface (J-AAPI) se ha propuesto como un estándar pare el desarrollo de agentes móviles basados en Java. Este API provee los mecanismos pare la creación, inicialización, manejo de mensajes, despacho, recuperación, activación, desactivación, generación de clones, o destrucción de agentes [Lange y Chang 1996].

El J-APPI esta integrado por lo siguiente:

Interfaces:

AgletContext. Genera un contexto en el cual el aglet es creado, desactivado, eliminado entre otras acciones. Es usada por un aglet para obtener información acerca de su ambiente y mandar mensajes hacia el ambiente, incluyendo otros agentes que actualmente se encuentran

activos en ese ambiente. Provee medios pare mantenimiento y administración de los aglets en ejecución. Los contextos son típicamente creados por el sistema, el cual tiene un demonio activado en cierto puerto, esperando la llegada de un nuevo aglet o para asistirlo en su proceso de migración. Los aglets que arriban son insertados en el contexto por medio del demonio [Lange y Chang 1996].

Message Manager

Future Reply

Clases:

Aglet. Está es una clase abstracta que se utiliza como clase base cuando se desarrolla un programa aglet. Esta clase define métodos para controlar el ciclo de vida del aglet, como son creación, clonación, despacho, retracción, desactivación, activación, destrucción, mensajes.

AgletIdentifier. A cada aglet se le es asignado un identificador único el cual es preservado por el aglet durante todo su ciclo de vida. Esta clase es una abstracción conveniente para su identidad. El objeto identificador esconde la representación específica de implementación de la identidad del aglet.

AgletProxy. Para la interacción entre Aglets, un aglet normalmente no invoca los métodos de otro aglet directamente. Para lograr eso hace *uso* de un *objeto AgletProxy, el* cual sirve como representante del aglet. La clase AgletProxy contiene métodos que permiten a los aglets hacer requerimientos sobre otros aglets para que estos realicen una acción. Estas acciones son dispatch(), clone(), deactive(), dispose().

Itinerary. Al proveer a un aglet con un itinerario se provee una forma apropiada para hacer viajes con múltiples destinos. Esta es una clase abstracta por tanto no puede instanciarse directamente. Así que debe crearse una subclase y heredar en esta la clase Itinerary.

Message. Una importante propiedad de los aglets es que se pueden comunicar. La comunicación entre agentes es soportada de modo que intercambiar mensajes, los agentes no necesariamente deben conocerse entre si. La principal forma de comunicación es a través del paso de mensajes. Los mensajes son objetos. Un objeto mensaje esta caracterizado por kind, una propiedad que es usada para distinguir los mensaj es . La clase message soporta un rango de constructores que tienen kinds como argumento.

Más detalles de AW13 se pueden encontrar en [Lange y Oshima 1997] y [Lange 1997].

Pérez Lezama, C. V. 1998. **Agentes Móviles en Bibliotecas Digitales**. Tesis Maestría. Ciencias con Especialidad en Ingeniería en Sistemas Computacionales. Departamento de Ingeniería en Sistemas Computacionales, Escuela de Ingeniería, Universidad de las Américas Puebla. Mayo.