Publicaciones en Ciencia y Tecnología **Vol 2N° 1** (2008), 23–34

INTERFACES DINÁMICAS EN DISPOSITIVOS MÓVILES.

[*]RODOLFO CANELÓN, ELIEMAR GÓMEZ, LEIVAN RIVERO

Resumen

En los últimos años se ha incrementado el uso de dispositivos móviles, los teléfonos celulares por ejemplo, han dejado de ser sólo instrumentos de transmisión de voz para convertirse en aparatos capaces de procesar y transmitir información. El desarrollo de aplicaciones para estos nuevos dispositivos debe tener en cuenta las peculiaridades de los mismos, las limitaciones en cuanto a la interfaz del cliente, ejecución de tareas, interacción de los datos entre otras. La finalidad de este trabajo es explicar por qué usar el patrón mediator como una posible solución al problema de las interfaces dinámicas implementándolo a través de los agentes móviles, asimismo incorporar un modelo de calidad para evaluar plataformas adecuadas para esta labor, la escogencia de los parámetros que definen el instrumento de evaluación es de vital importancia para este trabajo, finalmente la solución se aplica a un caso de estudio en el dominio del comercio móvil.

Palabras Clave: Agentes, Mediator, Patrones, Plataformas.

ADAPTIVE MOBILE USERS INTERFACES

Abstract

In the last years the use of movable devices has been increased, the cellular telephones for example, they have stopped being only instruments of transmission of voice to become apparatuses able to process and to transmit information. The development of applications for these new devices must consider the peculiarities of such, the limitations as far as the interface of the client, execution of tasks, interaction of the data among others. Purpose of this work is to explain why to use the mediator pattern like a possible solution to the problem of the dynamic interfaces being implemented it through the movable agents, also to incorporate a quality model to evaluate platforms adapted for this work, The selection of the parameters that define the evaluation instrument is of vital importance for this, finally the solution is applied to a case of study in the dominion of the movable commerce.

Keywords: Agents, Mediator, Patterns, Platforms

1. Introducción

Los grandes avances en microelectrónica y las redes de comunicaciones han hecho posible que los dispositivos móviles fortalezcan sus propiedades y cada día mejoren la prestación de servicios, en igual forma la masificación de los dispositivos móviles ha influido notablemente no solo en las organizaciones, sino también en el ciudadano común. Poder llevar un computador prácticamente a cualquier lugar y en cualquier momento (computing anytime and anywhere), está haciendo repensar la forma de hacer aplicaciones móviles. Para los desarrolladores este es un gran problema, pues deben crear las mismas aplicaciones pero en dispositivos más pequeños y con diferentes característica. De todas las propiedades de un dispositivo (capacidad de procesamiento, memoria limitada, conexión a red, tamaño...) la más importante es la que trata de movilidad. El trabajo que a continuación se presenta aporta una solución al problema de las interfaces dinámicas que se presentan en el contexto móvil. Se explica cómo el patrón mediator puede resolver el problema de las interfaces dinámicas y se propone una forma de implementarlo

[*] *Coordinación de Postgrado,Universidad Centroccidental "Lisandro Alvarado, Barquisimeto, Venezuela", Barquisimeto, Lara 3001*

a través de los agentes, distinguimos algunos patrones y cómo la solución que aportan ayuda en el desarrollo, pasamos a evaluar ciertas alternativas para el desarrollo de agentes, finalmente se crea una instancia de este modelo para el dominio comercio móvil.

2. El Patrón Mediator para solucionar el problema de interfaces dinámicas

Los dispositivos móviles son aparatos pequeños, con algunas capacidades de procesamiento, móviles o no, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras más generales. Estos dispositivos se pueden considerar ya como ordenadores personales con todo el significado de la palabra por lo que disponen de capacidad de procesamiento y de almacenamiento de datos [1]. Éstos por ser una tecnología relativamente nueva, en cierta forma carente de plataforma que le permita la ejecución de un programa (como abrir un documento de texto), en distintos dispositivos móviles, sin tener que realizar cambio alguno al programa. Igual pasa con la utilización de servicios disponibles en la red; generalmente deben descargarse e instalarse el componente de software realizado para determinada familia de móviles, para así poder utilizar el servicio que el usuario quiere ejecutar.

Por lo general, lo ideal es que el usuario, simplemente se conecte y use tales servicios. Debido a estos problemas de compatibilidad entre los dispositivos móviles y los servicios, se han ido desarrollando aplicaciones para solventar este problema. Estas soluciones tienen su fundamento en los patrones de diseño, como bases para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

En particular, las interfaces de usuario (IU) se diseñan en general adhoc para cada dispositivo específico. La heterogeneidad incrementa los costos de desarrollo de aplicaciones, sobre todo para el componente interfaz [2].

Retomando el punto sobre la interacción entre los dispositivos móviles y la red, se observa entonces que la idea es conseguir el componente asociado al móvil para que pueda éste funcionar. Una solución sería desarrollar un componente, que automáticamente buscara el software que sirva para ejecutar un determinado servicio, sin que el usuario note como se comunican. Esto conduce a buscar un patrón de comportamiento, que concuerde con la solución planteada anteriormente. Este patrón resulta ser el Mediator, puesto que su concepto, define un objeto que coordina la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.

El patrón ***M*ediator** permitirá disminuir el acoplamiento entre los móviles y los distintos servicios disponibles en la red, simplificando y centralizando el control de la comunicación entre éstos. Los componentes de este patrón son los siguientes:

- **mediador** (***M*ediator**): que define la interfaz de comunicación entre los colegas, permitiendo la cooperación entre ellos para la realización de tareas. Este componente conoce y mantiene a todos los colegas.
- **colegas *(*Collegue)**: sabiendo que cada uno de estos conoce tan solo a su mediador. Estos pueden hacer requerimientos únicamente a través de él.[6].

A continuación se describe el comportamiento de este patrón: Un colega (que actúa en este caso como cliente) hace un requerimiento al mediador correspondiente. Luego, el mediador busca en sus registros a aquellos colegas que pueden hacer frente a tal requerimiento, enviándoles entonces la información necesaria. El colega que acepta el requerimiento (actuando en este caso como servidor) no conoce al colega cliente, tan solo produce una respuesta apropiada y la envía al mediador, que toma la respuesta y la comunica al cliente que originalmente ha realizado la demanda del servicio. Es necesario señalar que mediador debe conocer a todos sus colegas en tiempo de compilación, por lo que se mantiene una estructura estática. Como ninguno de los colegas conoce la existencia de otros colegas, se logra de esta manera un

bajo acoplamiento, que permite efectuar cambios sin afectar los otros componentes [6]. Instanciando este modelo al problema de interaccin entre los dispositivos móviles y los servicios en la red, vemos que los colegas fungen como dispositivos móviles y el mediator traduce la solución planteada anteriormente.

3. Los Agentes Móviles como implementación del patrón Mediator

En la primera parte observamos cómo se comporta, en teoría, una posible solución al problema planteado, ahora veremos por qué usar agentes móviles como implementación del patrón mediator.

Antes de realizar la justificación comencemos por definir agentes. Los Agentes de software son entidades que pueden actuar en nombre de otra entidad y que poseen ciertas propiedades deseables como autonomía, flexibilidad (Reactividad-Adaptabilidad, Proactividad, Comunicabilidad-Sociabilidad), continuidad temporal, movilidad (Agentes móviles), capacidad de Razonamiento / Aprendizaje y Seguridad / Confiabilidad. [8].

Los agentes a su vez se subdividen en dos grandes grupos: los agentes estáticos (proceso que se ejecuta única y exclusivamente en la máquina donde comienza la ejecución) y los agentes dinámicos (proceso que no est limitado al sistema donde se inicia la ejecución sino que tiene la completa libertad para moverse por la red, migrando, interactuando y regresando bajo su propio control), siendo este último grupo el que despierta mayor interés, debido a que pueden ejecutarse en distintos dispositivos y tienen la capacidad de moverse en la red.

Con respecto a los agentes móviles, es precisamente la propiedad de la migración la que les da la capacidad de movimiento. Este movimiento se realiza de la siguiente forma: cuando un agente se mueve, realiza una migración suspendiendo la ejecución, transportándose a otra plataforma en la red y reanudando la ejecución en el punto en que se suspendió. En este contexto, existen dos tipos de migración que se diferencian por el transporte de más o menos información o datos de estado, que son:

- Migración fuerte: Transporta el estado del objeto (valores de los atributos del agente) y estado de la máquina o estado de ejecución (contador de programa, punteros de pila, registros ...). Es la foto completa, justo cuando se suspende la ejecución para su posterior reanudación en el entorno de otra máquina.

- Migración débil: Transporta sólo el estado del objeto (valores de los atributos del agente). Es la foto incompleta, que permite sólo reiniciar la ejecución con los valores actuales de los atributos. Por otro lado, es la típica migración utilizada por los agentes móviles escritos en Java, debido a que este lenguaje no permite la captura del estado de ejecución de un hilo de Java. Se resalta que el modelo de seguridad de la arquitectura JVM (Java Virtual Machine) impide que un programa acceda o manipule directamente sus punteros, registros y pilas. Por consiguiente, cuando un agente Java se transporta de una plataforma a otra, no reanuda su ejecución sino que reinicia ésta con los valores actuales de sus atributos [8].

Para implementar el patrón mediator a través de los agentes, se necesita de un componente, que automáticamente, pueda dar comunicación entre el dispositivo móvil y el servicio que se requiere en determinada red, es aquí donde los agentes móviles juegan un rol importante, puesto que son autónomos y tienen la capacidad de moverse en la red. De este modo se podría usar un agente que controle la comunicación entre el móvil y el servicio, de tal forma, que si el móvil necesita cierto componente, para una familia definida, ste se encargue de buscar los datos necesarios para que el servicio se pueda ejecutar, y así este proceso no lo ejecutaría el usuario.

4. Patrones de Diseño en Agentes Moviles

Se han considerado tres clases de patrones fundamentales [7] para el diseño de agentes que son: *migración (traveling), tarea e interacción.* Éste esquema de clasificación hace más fácil el entender el dominio y aplicación de cada patrón, el distinguir los diversos patrones y el descubrir nuevos patrones.

4.1. Patrones de Migracin

La migración es la esencia de los agentes móviles. Los patrones de migracin tratan sobre diversos aspectos de la gestin de movimientos de los agentes mviles, como lo son el enrutado (routing) y la calidad de servicio. Estos patrones nos permitirn reforzar la gestin de la movilidad, entre este tipo de patrones se mencionan los siguientes:

- **Itinerario**: proporciona una manera de ejecutar la migración de un agente, que será responsable de realizar una tarea dada en anfitriones remotos. El agente recibe un itinerario de la agencia fuente, indicando la secuencia de agencias que debe visitar. Una vez en la agencia indicada, el agente ejecuta su tarea localmente y después continúa en su itinerario. Luego de visitar la última agencia, el agente va de regreso a su agencia fuente. La Figura 1 ilustra este patrn. [3].

- **Envío (forward)**: provee una forma para que un servidor envíe automáticamente a un agente que acaba de llegar a otro servidor.

- **Acceso (ticket)**: especifica una dirección destino y encapsula la calidad del servicio y permisos requeridos para enviar a un agente a la dirección del servidor y entonces ejecutarse ahí.

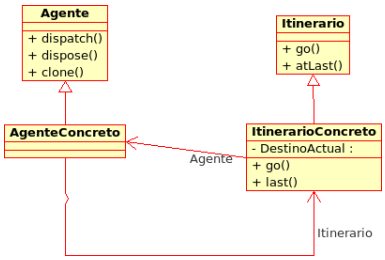

Figura 1: Patrón Itinerario

En el presente trabajo el patrón de *itinerario* es el que se adapta a la solución planteada debido a que la clase *itinerario* provee una interfaz para mantener actualizado el itinerario de los agentes y despacharlos a nuevos destinos, asimismo es ampliamente utilizado: Para Proveer una interfaz uniforme en el traslado de agentes de forma secuencial a múltiples servidores. Para definir recorridos que pueden ser compartidos por varios agentes. [7].

Estos servirá para agilizar las búsquedas en la red de algún requerimiento, cuando el usuario o el móvil solicitan un servicio en esta, ya que tendrá una lista de servidores a donde irá, reconocerá si existe o no ese servidor y además se podrán reutilizar las listas utilizada por otros agentes.

4.2. Patrones de Tareas

Los patrones de tareas se refieren al análisis y delegación de tareas, las cuales se pueden asignar a un agente dinámicamente o a un grupo de agentes para llevar a cabo un trabajo en paralelo. Existen dos patrones de tareas fundamentales:

- **Maestro-Esclavo**: define un esquema con el cual un agente maestro puedo delegar una tarea a un agente esclavo. El agente esclavo se desplazada al servidor especifico, ejecutar la tarea y regresar con el posible resultado.

- **Plan**: provee una forma de definir la coordinacin de mltiples tareas, para ser ejecutadas en mltiples servidores secuencialmente o en paralelo por varios agentes. El plan encapsula el flujo de tareas y promueve su reutilizacin as como la asignacin dinmica de las mismas. [7].

Como se observa en la figura 2, el propósito general del patrón maestro-esclavo es crear clases abstractas *Maestro y esclavo*, para localizar las partes invariantes de la delegación de tareas. Los agentes maestros y esclavo son definidos como subclases de *Maestro y Esclavo*, en los cuales sólo varía la parte de la tarea que ejecutará y cómo el agente maestro debe gestionar el resultado de la tarea implementada

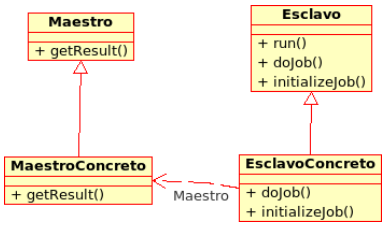

Figura 2: Patrón Maestro Esclavo

Este patrón provee una forma fundamental de reutilizar código entre clases de agentes y se usará principalmente por los siguientes aspectos:

- Cuando un agente necesita desempeñar una tarea en paralelo con otras tareas de las cuales es responsable.

- Cuando un agente estacionario quiere desempeñar una tarea en un destino remoto. [7]

4.3. Patrones de interacción

Los patrones de interacción se refieren a la localización de agentes y a la facilitación de sus interacciones [9]. Así pues, en este grupo se resaltan los siguientes:

- **Reunión (Meeting)**: sugiere una manera de promover interacciones locales entre los agentes distribuidos en la red. Tales interacciones hacen posible la ejecucin de tareas dadas, así como la optimización de resultados. [3]

- **Locker**: define un espacio de almacenamiento específico para que un agente deje datos antes de ser enviado temporalmente a otro destino y pueda evitar el trasladar datos que no va a utilizar en ese momento.

- **Mensajero**: define un agente subordinado para llevar un mensaje de un agente a otro, mientras el agente principal contina con su tarea.

- **Ayudante (facilitator)**: Define un agente que provee servicios de nombrado y localizacin de agentes con capacidades especficas. [3].

La implementación del patrón Reunión (Meeting) se muestra en la figura , allí se observa que aquellos agentes que necesiten interactuar unos con otros estarn equipados con un objeto Reunión. Ellos usarán un identificador único para localizar un objeto *GestorReunión* para registrarse así mismo. Luego el objeto gestor de reunión notificará a los agentes que ya se encuentran registrados de la llegada del nuevo agente,

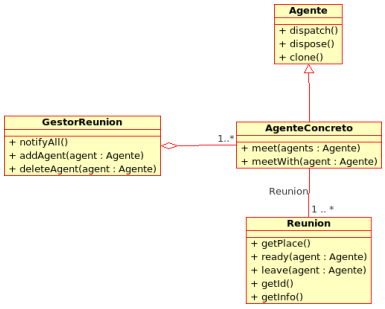

Figura 3: Patrón Reunión (Meeting)

éste obtendrá una referencia y viceversa para que puedan iniciarse las interacciones. Los agentes deberán borrar su registro por ellos mismos antes de abandonar el lugar de reunión. Debido a que poseen un único identificador se pueden ejecutar varias reuniones simultáneamente en el mismo servidor.

Este patrón se aplica cuando se requiere que los agentes de diferentes servidores interactúen en un mismo sitio localmente; en los siguientes casos:

- Cuando los agentes necesitan interactuar y el costo de viajar a un lugar común es menos que el asociado con una comunicación remota.
- Cuando los agentes necesitan acceder a servicios locales en un servidor específico [7]

5. Plataforma de Desarrollo

Sobre la base de las consideraciones anteriores, queda claro que para resolver el problema se necesita construir un sistema multiagente (SMA).

En el mercado, existe un amplio número de propuestas, herramientas, y plataformas de diferente calidad y madurez (más de 100 productos de software) que probablemente satisfacen los requerimientos de nuestra investigación, pero por razones de tiempo y costo sólo se evaluarán algunas.

Cada plataforma de agentes ser evaluada bajo los siguientes requisitos:

- **Portabilidad de la plataforma**: Si es portable a dispositivos mviles y bajo distintos sistemas operativos.
- **Facilidad de desarrollo**: debe proporcionar los servicios básicos comunicaciones, identidad, movilidad que deben estar presentes para que los Agentes puedan ejecutarse.
- **Documentación y Soporte**: se refiere a si la documentación es completa y es suficiente para desarrollar una aplicación y en cuanto al soporte si existen foros, comunidades y ayudas que se puedan utilizar.
- **Versiones actualizadas**: Si la versión que está es reciente y si se siguen publicando nuevas versiones, es muy importante conocer si se trata de una versión estable, y las mejoras realizadas en versiones anteriores.
- **Interacción con interfaces gráficas de usuario**: Facilidad de implementarlas y de poder interactuar con ellas por medio de agentes sin tener que acoplarlos.
- **Uso de protocolos estándar**: Utilicen los protocolos del mercado y no uno diferente que pueda presentar problemas frente a un firewall al utilizar un puerto no conocido.

- **Facilidad para la movilidad en cualquier tipo de red**: Si el agente se puede mover de una LAN a una red inalámbrica, por ejemplo.
- **Facilidad de instalación**: Si la plataforma se puede instalar en distintos lugares sin muchas complicaciones.
- **Manejo óptimo de recursos de hardware y software**: Esto con el fin que no sean un impedimento en los dispositivos móviles.
- **Extensibilidad de la plataforma**: Que tan factible es seguir extendiendo la plataforma sin ningún problema.
- **Disponibilidad de librerías**: Que tantas opciones brinda el API para desarrollar de distintas maneras.
- **Independencia de proveedores**: Que tanto se está ligado a los proveedores y si las consecuencias de un cambio son muchas.
- **Seguridad**: que tan segura es la plataforma con respecto a los agentes y a la plataforma misma. Se tiene en cuenta si el agente es encriptado al viajar o si la plataforma maneja autorizacin y autenticación.
- **Calidad de la arquitectura de agentes**: Si la plataforma ofrece una arquitectura de agentes y que tan robusta es.
- **Soporte SMA**: Permite distribuir el SMA en diversos ordenadores (con distintos S.O.) y controlar la configuración mediante una interfaz gráfica.
- **Soporte de Patrones**: soporta el uso de patrones, y mas aun que pueda ofrecer formas de implementarlos.
- **Compatibilidad Estándar**: los estándares comunes para tecnología de agentes son FIPA, OMG (MASIF) entre otros. Algunas de las plataformas evaluadas en este informe vienen de la recomendación FIPA para implementaciones, otras son evaluadas debido a su popularidad general entre usuarios/desarrolladores.

A continuación las caractersticas más importantes de cada una de ellas:

5.1. JadeLeap:

LEAP se desarrolló como una API que se debe usar dentro del ambiente de JADE para su ejecucin en ambientes móviles. Inicialmente el proyecto se enfocó en desarrollar ambientes con plantillas reducidas y en la compatibilidad con los entornos móviles de Java (J2ME). Luego, se implemento todo un ambiente con suficientes herramientas para los desarrolladores y compatible con todos los ambientes de Java : "JADELEAP" JADELEAP puede ser implementado sobre diversos tipos de dispositivos móviles, redes y JVMs, con un conjunto homogéneo de APIs, permitiendo el desarrollo de aplicaciones basadas en agentes para ambientes móviles [5].

5.2. Aglets:

Fue desarrollado por IBM Japón y su distribución es gratuita. La palabra Aglet se compone de la palabra agente y Applet. La intención es guardar una similitud entre los agentes de software y los Applets de Java. Un Aglet es simplemente una clase en Java que extiende de Aglet, además en una forma muy básica, utiliza muy poco del framework. El framework del ASDK (Aglets Software Development Kit) es bastante extenso.

En adición a los métodos por defecto de la clase Aglet (onCreation, onDisposing, run), existen otros para adicionarle listeners como, addMobilityListener. Además, el framework provee un modelo de eventos que soporta operaciones orientadas a movilidad y triggers. Un Aglet puede ser despachado y devuelto de contextos remotos y en cualquiera de ellos puede efectuar una operación permitida por Java y por las políticas de seguridad configuradas en ese sistema.

5.3. BESA

La plataforma BESA (Behavior Oriented, Event Driven and Social Based Agent) combina los conceptos de sistemas multiagentes, con el diseo de sistemas concurrentes. Su origen es a partir del proyecto ASMA (Arquitectura para Sistemas Multi Agentes) desarrollado en la Pontificia Universidad Javeriana. La arquitectura BESA ha sido implementada completamente en Java, para permitir la portabilidad entre diferentes plataformas. Su entorno es orientado a objetos, soporta multithreading y tiene gran facilidad en las comunicaciones.[10]

5.4. J2ME

Java 2 Micro Edition es una plataforma de desarrollo de aplicaciones para dispositivos con recursos limitados tanto en pantalla gráfica como en procesamiento y memoria, tal es el caso de teléfonos móviles, PDAs, Handhelds, Page, electrodomésticos inteligentes, etc. Actualmente las compañías telefónicas y los fabricantes de móviles están implantando los protocolos y dispositivos para soportar esta plataforma. La plataforma J2ME se puede dividir en varios componentes que forman la arquitectura, estos componentes son:

- La Java Virtual Machine con diferentes requisitos, cada una para diferentes tipos de dispositivos.
- Maneja un sistema de configuraciones las cuales se definen como un conjunto mínimo de APIs que agrupan el desarrollo para diferentes dispositivos.
- Adiciona un grupo de perfiles, que se encargan de controlar el ciclo de vida de las aplicaciones, interfaz de usuario, etc.

La tabla 1 muestra la escala para evaluar las plataformas de acuerdo a los requerimientos mencionados

Caracteristícas	Definición	Puntaje
No soporta	No apoya el Requisito	1
Poco Soporte	De cierta manera se puede satisfacer el requisito o parte de él	2
Fuerte Soporte	Este requisito aparece explícitamente en la descripción de la plataforma o en su manual.	3

Cuadro 1: Escala de evaluación

En la evaluación se observa que las opciones tecnológicas con más oportunidad de elección son JadeLeap y J2ME. Se selecciona JadeLeap por su facilidad, a la hora de desarrollar los agentes, puesto que ofrecen interfaces gráficas, adems de presentar una seguridad mas aceptable que J2ME, una buena disponibilidad de librerías y una excelente calidad de la arquitectura de agentes, lo que hace presagiar que el desarrollo de agente en esa plataforma ser de cierta forma rápida.

Criterio	Aglets	Jade-Leap	Besa	J2Me
Facilidad para implementar movilidad.	3	2	2	2
Portabilidad de plataforma sobre dispositivos móviles	1	3	1	3
Facilidad de Desarrollo	1	2	2	3
Documentación y Soporte	1	3	3	3
Versiones Actualizadas	1	3	3	3
Interacción con interfaces gráficas de usuario	2	3	2	2
Uso de protocolos estándar.	2	2	2	3
Facilidad para la movilidad en cualquier tipo de red	3	2	1	2
Facilidad de instalación	3	3	3	3
Manejo óptimo de recursos de hardware y software	2	3	2	3
Extensibilidad de la plataforma.	1	2	2	3
Disponibilidad de librerías.	3	2	2	1
Independencia de proveedores	2	2	1	2
Seguridad.	3	2	2	1
Calidad de la arquitectura de agentes	2	3	3	1
Soporte SMA	1	3	2	

6. Caso de Estudio

A continuación se muestra con un breve ejemplo de cómo se puede resolver el aspecto de la interoperabilidad en el contexto de una aplicación basada en agentes heterogéneos. En el caso de estudio se utiliza un modelo de negocios en el cual los compradores acceden a los productos ofrecidos por un vendedor, en base en sus requerimientos. Está definida en dos sistemas, el Sistema del Comprador y el del Vendedor.

6.1. Sistema del Comprador

El sistema del comprador está conformado por la interfaz gráfica de usuario, la plataforma de agentes, el componente de seguridad y el componente de datos. (Ver figura 4)

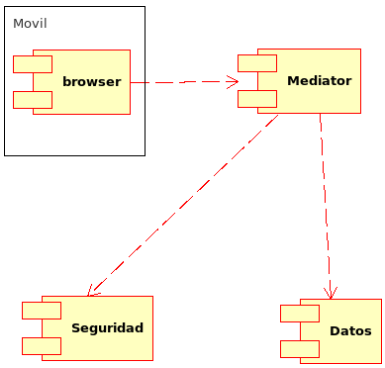

Figura 4: Sistema del Comprador

El sistema del comprador se encuentra instalado en un equipo denominado "Comprador", al cual acceden los usuarios compradores por medio de la interfaz de usuario. Un comprador puede acceder al sistema desde cualquier dispositivo móvil que posea conexión a la red. Los usuarios, sólo necesitan un navegador para Internet, mediante el cual se conectan al servidor y usan el sistema A continuación se hace una descripción de cada uno de ellos.

- **Interfaz Gráfica de Usuario (GUI)**: Permite especificar al usuario la información para ingresar al sistema, esta información corresponde a los requerimientos, preferencias sobre los artículos y demás datos que sean necesarios. Además se utiliza como medio para mostrar al usuario la informacin que los agentes han encontrado.
- **Plataforma de Agentes**: En ella se encuentra el Sistema de Agentes Estacionarios y los agentes móviles. Los agentes estacionarios son aquellos que permanecen en la máquina del comprador y cumplen funciones como el despacho de agentes a los sitios de los vendedores y la recepción de los agentes móviles cuando regresan con la información que han recolectado. Los Agentes Móviles son agentes creados para la búsqueda y compra de artículos, llevan la información de las preferencias del usuario sobre los artículos que desea y el itinerario con los sitios a visitar.
- **Componente de Seguridad**: Este componente est subdividido en dos módulos, el primero se encarga del cifrado y descifrado de los datos y el segundo de los procesos de firmado y verificación de firmas. Este componente es utilizado por el Agente Despachador para solicitar el cifrado y firmado de la informacin que va a transportar el agente móvil. También es usado por el Agente Recepcionista para verificar la firma del vendedor y descifrar la informacin que se recibe del agente móvil.
- **Componente de Datos**: Este componente est conformado por la Base de Datos y el Componente Administrador de la Base de Datos. La primera contiene una lista de direcciones de los sitios de los diferentes vendedores, así como el tipo de arículos que ofrecen, este listado es usado para la generación del itinerario del agente móvil; también se almacena la información relacionada con las solicitudes del comprador y el resultado de los procesos realizados por el Agente Móvil. El Componente Administrador de la Base de Datos es el encargado de realizar las consultas y modificaciones a la información almacenada en la base de datos.

6.2. Sistema del Vendedor

Para ingresar al Sistema del Vendedor, el usuario se conecta a través de la red a la GUI. Por razones de Seguridad el usuario debe utilizar cifrado/descifrado de mensajes, así como también firmas de seguridad. Una vez que sea válida la entrada, el Usuario puede hacer búsquedas en la red de los artículos que quiere. El sistema mediante sus agentes móviles toma la información de los artículos y las preferencias del Usuario, como puede ser el tiempo de entrega del artículo, costo, etc., cuando se tienen todos los datos los agentes se mueven a través de la red a cumplir con la búsqueda de información y cuando la hayan recolectado regresan a su sitio de origen para reportar los resultados obtenidos.

6.3. Itinerario de un Agente Móvil en el sistema del comprador

El proceso que cumple el agente móvil del Comprador, siguiendo su itinerario, consiste en el desplazamiento de su código desde el Sistema del Comprador a cada uno de los sitios de los vendedores. Cuando llega, interactúa con los agentes del Vendedor y con base en las preferencias de su usuario procede a realizar las tareas encomendadas.

Uno de los posibles campos de aplicación de los agentes móviles es el comercio electrónico; en la actualidad hay disponibles sistemas basados en agentes móviles para Comercio Electrónico, se cuenta con estándares, plataformas y productos que facilitan su desarrollo, pero aún el comercio electrónico no ha llegado al auge esperado, una de las razones para que esto suceda son los problemas de seguridad existentes.

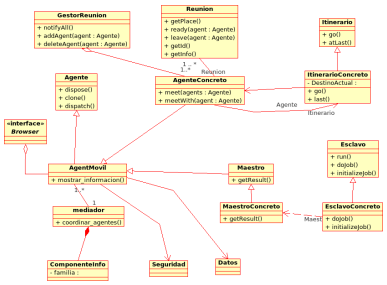

Figura 5: Diagramas de Clase de la Posible Solución

Por esos problemas se hace necesario buscar opciones para disminuir el riesgo al realizar transacciones de comercio electrónico y la manera de lograrlo es implementar los siguientes aspectos de seguridad: Autenticación, autorización, Norepudio, Confidencialidad e Integridad.

A través de la GUI el usuario puede ver el listado que contiene los resultados que trajeron los agentes, y con base en esa información si lo desea da la orden de crear un agente móvil comprador que viaje al sitio elegido y procese la orden de compra.

7. Conclusión

En el siguiente trabajo se implementa una solución al problema de Interfaces Dinámicas usando el patrón mediator implementado con agentes bajo la plataforma Jade. En el mismo se presenta patrones de diseño para agentes clasificándolos en patrones de migración, patrones de tareas y patrones de interacción. Se usan agentes dinámicos, debido a que pueden funcionar en distintos dispositivos y tienen la capacidad de moverse en la red. Un aspecto a resaltar es la plataforma de desarrollo, ésta se evalúa dentro de un rango de propuestas de diferente calidad y madurez. Para la escogencia de la plataforma de desarrollo se presenta un modelo de evaluación, los aspectos a considarar son definidos y se aplica el instrumento de forma tal que conduzca a la plataforma idónea para implementar la solución, en este caso JadeLeap una API que se desarrolló para ser usada dentro del ambiente de JADE en ambientes móviles, resultó ser la plataforma selecionada. Finalmente se expone un caso de estudio, donde se muestra el la interación de los agentes, basados en el patrón mediator.

Referencias

[1] Calero Antonio (2005) *"Tecnología móviles con JAVA"*, Revista del Instituto Tecnológico de informática, Actualidad (04)

[2] Canelon R., Losavio F., Matteo A.(FEB. 2007), *"Modeling Quality of adaptive mobile user interfaces"*, Revista de la Facultad de Ingenería U.C.V VOL 22, N1, PP 4559.

[3] Ferreira Emerson, Duarte Patrícia, Abrantes, Ronisonflavio,(2003, NOV) *"Implementing Mobile Agent Design Patterns in the jade Framework"*

[4] Fonnegra N., Rusii M. (2005 FEB.) *"Plataforma de Servicios distribuidos basados en agentes móviles"*.

[5] García A., Solarte Z J. (2005, MARZO). *"Agentes en computación móvil"*, Universidad Autónoma de Occidente.

[6] Losavio F. , Guillen. C (2007, SEP), *" Diseño Arquitectónico de Sistemas Distribuidos en rapide1*

[7] Perez D., Jesus A. (2000, ABR). *"Agentes Moviles: Programación, Seguridad y diseño"*, Universidad de Oviedo.

[8] Yaguez J. (2007, SEP). *" Arquitecturas y Middleware de Comunicaciones"* [En Línea]. Disponible en:

[9] Yariv Aridor, Danny Lange (MAY. 1998) *Infrastructure for mobile agents"*

[10] Abrantes Ronisonflavio *"Implementing Mobile Agent Design Patterns in the Jade framework"*, Proceding in the 2ND International Conference in Autonomus Agent.