PUMAS: Un Framework que Adapta la Información en Ambientes Ubicuos

Angela Carrillo Ramos[1], Jérôme Gensel, Marlène Villanova-Oliver, Hervé Martin
LSR-IMAG Laboratory, SIGMA Team. B.P. 72
38402 Saint Martin d'Hères Cedex, France
{carrillo, gensel, villanov, martin}@imag.fr

Resumen

Los *Dispositivos Móviles* (*DM*) se caracterizan por sus intrínsecas capacidades reducidas (por ejemplo, tamaño de la pantalla, memoria, capacidad de almacenamiento). *PUMAS* (*Peer Ubiquitous Multi-Agent System*) es un *framework* basado en agentes cuyo principal objetivo es proveer a usuarios nómadas con información adaptada a diferentes criterios (por ejemplo, sus preferencias, localización, etc.) cuando acceden *Sistemas de Información Web* (*SIW*) a través de sus *DM*. Los *SIW* pueden ejecutarse en uno o más servidores o, en uno o más *DM*. Los agentes de *PUMAS* están organizados para llevar a cabo una búsqueda *inteligente* (basada tanto en el conocimiento propio, adquirido e inferido de los agentes, como en su capacidad de razonamiento) y, *adaptativa* (ya que toma en cuenta las características del usuario y las de su *DM*) de la información que el usuario requiere proveniente de uno o más *SIW*. Cuando un usuario nómada busca información a través de su *DM*, sus consultas se propagan a través de los agentes de *PUMAS* (que adicionan a estas consultas algunas características del usuario y de su *DM* para ser tenidas en cuenta al adaptar la información) hacia el *Router Agent* (del *SMA* de *Información*). Dicho agente ejecuta el proceso de enrutamiento de consultas (*Query Routing*) que consiste en la búsqueda de fuentes de información adecuadas que pueden responder las consultas del usuario teniendo en cuenta las preferencias del usuario, las características de su *DM*, su localización, etc. Esta búsqueda se basa en el conocimiento manejado e intercambiado por los agentes de *PUMAS* (especialmente, aquellos pertenecientes a los *SMA* de *Información* y de *Adaptación*). Este artículo presenta el conocimiento manejado e intercambiado por los agentes de *PUMAS* y la forma en que se lleva a cabo el proceso de *Query Routing* en tres etapas: análisis de la consulta, envío de la misma y recepción de sus resultados.

Palabras claves: *Sistemas Multi-Agente, Computación Ubicua, Adaptación, Enrutamiento de Consultas, Recuperación de información***.**

Abstract

Mobile Devices (*MD*) are characterized by intrinsic reduced capacities (e.g., size of screen, memory, data storage). *PUMAS* (*Peer Ubiquitous Multi-Agent System*) is a framework based on agents whose main objective is to provide nomadic users with information adapted to different criteria (their preferences, their location, etc.) when they access *Web-based Information Systems* (*WIS*) using such *MD*. *PUMAS agents* are organized in order to achieve an *intelligent* and *adaptive* information search. This search is on the one hand *intelligent* because is based on the knowledge of the agent (proper, acquired and inferred knowledge) and its capability of reasoning and, on the other hand, is *adaptive* because it takes into account the nomadic user's profile, characteristics of her/his *MD* and ubiquitous context features (e.g., location, connection time, etc.). When a user searches for information through her/his *MD*, her/his queries are propagated through the agents of *PUMAS* (which add into these queries some characteristics of both the user and her/his *MD* for adaptation purposes) towards the *Router Agent* (which belongs to the

[1] La autora *Angela Carrillo Ramos* cuenta con financiamiento parcial de la Universidad de los Andes (Bogotá, Colombia).

Information MAS of *PUMAS*). This agent performs a *Query Routing* process consisting in searching of the right information sources which can answer these queries taking into account the user's preferences, the features of her/his *MD*, her/his location, etc. This search is based on the knowledge managed and exchanged by the *PUMAS* agents (especially, those belonging to the *Information* and the *Adaptation MAS*). In this paper, we describe the different kinds of knowledge managed by the *PUMAS* agents and the way *PUMAS* performs the *Query Routing* process in three steps: the analysis of the query, the selection of the information sources and the redirection of the query.

Keywords*: Multi-Agent Systems, Ubiquitous Computing, Adaptation, Query Routing, Information Retrieval.*

1 Introducción

Las aplicaciones que se ejecutan sobre *Dispositivos Móviles* (*DM*) son diseñadas para permitir a los usuarios la consulta de datos en cualquier momento desde cualquier lugar. Esta es la idea de base de la *Computación Ubicua* (*Ubiquitous Computing*) definida por la *W3C* [13] como un paradigma emergente de computación personal que se caracteriza por el tipo de dispositivos de acceso utilizados (dispositivos de cómputo pequeños, inalámbricos, que se manipulan fácilmente con una o dos manos). Estos dispositivos requieren arquitecturas de red capaces de soportar su configuración automática y *"ad-hoc"* que tome en cuenta las características del *ambiente* en el que se desarrolla la *computación ubicua* (*ubiquitous computing environment*) tales como heterogeneidad, movilidad, autonomía, alta distribución, etc. Pirker *et al.* [10] definen tal ambiente como una red dinámica de sistemas y dispositivos embebidos que pueden interactuar con el usuario, con el fin de satisfacer sus requerimientos y de proveerlos con una variedad de servicios colaborativos de información y de comunicación.

Koch *et al.* [6] muestran cómo la *computación ubicua* se fortalece cuando la aplicación tiene la inteligencia para procesar información contextual (es decir, localización, tiempo de conexión, etc.) acerca del usuario y de su contexto, con el fin de proveerlo con la información adecuada en el momento adecuado, y adicionalmente, ofrecer la posibilidad de tomar cierta iniciativa en representación del usuario (características *proactivas* y *adaptativas*[2]). Estos autores recomiendan el uso de la *tecnología agente* en *aplicaciones ubicuas* ya que un sistema de cómputo móvil debe en primer lugar, tener la habilidad de ser *proactivo*, es decir, que razone sobre las actividades del usuario y analice cómo éstas pueden ser ejecutadas, y en segundo lugar, ser *adaptativo* debido a que el *ambiente* de *computación ubicua* es altamente dinámico (es decir, los usuarios pueden desplazarse de un lugar a otro y sus actividades pueden cambiar basadas en sus características contextuales tales como su localización, momento de conexión, actividades desarrolladas, etc.).

Con el fin de proveer al usuario nómada sólo con la *información más relevante* (es decir *"la información adecuada, en el lugar y momento adecuados"*), una aplicación que se ejecuta en *DM* debe contar con mecanismos para propagar las consultas (*queries*) de los usuarios hacia las fuentes de información *"adecuadas"* (almacenadas en uno o más dispositivos) que puedan responder estas consultas teniendo en cuenta las preferencias del usuario, las características de sus *DM*, su localización, etc. Este es el principal propósito del *enrutamiento de consultas* (*Query Routing*). Xu *et al.* [11] lo definen como el problema general de evaluar una consulta usando las fuentes de información más relevantes y de integrar los resultados provenientes de diferentes fuentes de información. Para optimizar el proceso de *enrutamiento de consultas*, los trabajos presentados en

[2] Koch *et al.* [6] definen *Proactividad* (*Proactivity*) como la habilidad computacional para anticipar las intenciones del usuario y otros eventos externos con el fin de actuar acorde a ellos. La *Adaptación* (*Adaptation*) es la habilidad para ajustar automáticamente el comportamiento del sistema a los cambios de las circunstancias.

[1] y [9] proponen el uso de ciertas métricas relacionadas con la confianza de las fuentes de información, su capacidad para satisfacer las necesidades de información de los usuarios y los tiempos de respuesta.

En [3], presentamos *PUMAS* (*Peer Ubiquitous Multi-Agent System*), un *framework* cuyo objetivo principal es el de recuperar de diferentes fuentes, la información que se ajuste tanto al perfil del usuario (el cual contiene sus necesidades, características, preferencias, historia en el sistema, localización actual, etc.), como a las características técnicas del *DM* que el usuario usa para acceder el *Sistema de Información Web* (*SIW*). La arquitectura de *PUMAS* se compone de cuatro *Sistemas Multi-Agente* (*SMA*), cada uno compuesto de uno o más agentes ubicuos (*ubiquitous agents*) que cooperan con el fin de llevar a cabo los diferentes servicios ofrecidos por *PUMAS* (tales como conexión/desconexión de los *DM*, búsqueda de información en diferentes fuentes, etc.). En *PUMAS* se utilizan archivos *XML* para representar la información manejada por los agentes (datos sobre las características del usuario, sus sesiones, su *DM* y sus preferencias, los roles de los agentes y las reglas que controlan su comportamiento).

Este artículo tiene como puntos centrales, por una parte, la representación del conocimiento manejado por los agentes de *PUMAS* con el fin de llevar a cabo la adaptación de la información, y por otra parte, el proceso de *enrutamiento de consultas* (*Query Routing*) ejecutado por el *Router Agent* (agente que pertenece al *SMA* de *Información* y cuyo rol se define en este artículo). Dicho proceso consiste en redirigir las consultas formuladas por el usuario hacia diferentes *SIW* (que se ejecutan en servidores o *DM*). Mostramos aquí cómo se usan en este proceso las *Bases de Conocimiento* (*BC*) manejadas por los agentes de *PUMAS*.

La estructura de este artículo es la siguiente: En la sección 2 presentamos algunos trabajos relacionados con *PUMAS* que consisten en arquitecturas que adaptan la información al usuario. Introducimos la arquitectura de *PUMAS* en la sección 3. Luego en la sección 4, mostramos algunos escenarios en los que se describe la conexión de un usuario a *PUMAS* así como el envío de una consulta y la posterior recepción de resultados. En la sección 5 describimos *piezas de conocimiento* (que en este trabajo llamamos *"hechos"*) utilizadas por los agentes de *PUMAS* para adaptar la información a las características del usuario y de su *DM*; mostramos particularmente los hechos utilizados por los agentes de los *SMA* de *Información* y de *Adaptación*. Después en la sección 6, mostramos un ejemplo del uso de nuestro *framework* en un *SIW* de un hospital. En la sección 7, describimos el proceso de enrutamiento de consultas (*Query Routing*) ejecutado por el *Router Agent*. En la sección 8 discutimos el impacto que pueden producir los cambios de localización en dicho proceso, particularmente, en la composición de la *"red de vecinos"*. Finalmente, exponemos las ideas sobre la implementación de *PUMAS*, antes de concluir.

2 Trabajos relacionados

En esta sección presentamos algunos de los trabajos cercanos al nuestro que consisten en arquitecturas o frameworks basados en agentes que adaptan la información a los usuarios.

CONSORTS [7] es una arquitectura basada en agentes ubicuos diseñada para el soporte masivo de *DM*. Dicha arquitectura detecta la localización del usuario y define su perfil a través de un *"Spatio-Temporal Reasoner"* (*STR*). *CONSORTS* utiliza la localización del usuario y su perfil para adaptarle la información. Esta arquitectura propone mecanismos para definir las relaciones entre agentes (es decir, comunicaciones, jerarquía, definición de roles), con el propósito de satisfacer las necesidades de información del usuario. Sin embargo, *CONSORTS* no considera aspectos como la distribución de información entre diferentes dispositivos (que podría mejorar tiempos de respuesta), ni la adaptación de la información a los dispositivos de acceso (*DM*). Tampoco considera las preferencias del usuario.

El trabajo de Gandon *et al.* [4] propone una arquitectura basada en *Semantic Web* que tiene en cuenta las características contextuales del usuario y su privacidad. Esta arquitectura soporta el

descubrimiento automático y el acceso a los recursos personales del usuario, sujetos a sus preferencias de privacidad. Los recursos personales y públicos son representados como *Servicios Web* (*Web Services*). Las reglas de invocación de servicios se definen a través de ontologías y de perfiles que permiten identificar las fuentes de información más relevantes para responder las consultas del usuario. Sin embargo, este trabajo no tiene en cuenta que la información necesaria para responder las consultas del usuario puede estar distribuida en los *DM*.

PIA [2] es un sistema de información personal basado en agentes que recolecta, filtra e integra información en un punto común, ofreciendo acceso a la información a través de *WWW*, e-mail, *SMS*, *MMS* y clientes *J2ME*. Combina técnicas de *push* y *pull* que permiten al usuario, por una parte, buscar explícitamente información específica y por otra parte, ser informado automáticamente sobre información relevante que está dividida en grupos (el usuario especifica su tiempo de trabajo y el sistema divide el día en pre, trabajo y recreación). Un agente personal maneja la información individual de cada usuario, brindándole información adaptada a su perfil. Cuenta con un mecanismo de retroalimentación para el perfil de usuario. Sin embargo, el sistema *PIA* solamente busca información en formato texto (es decir, documentos). No toma en cuenta la adaptación de la información a diferentes tipos de *DM* ni la localización del usuario.

3 El framework PUMAS

La arquitectura de *PUMAS* [3] está compuesta de cuatro *Sistemas Multi-Agente* (ver Fig. 1):

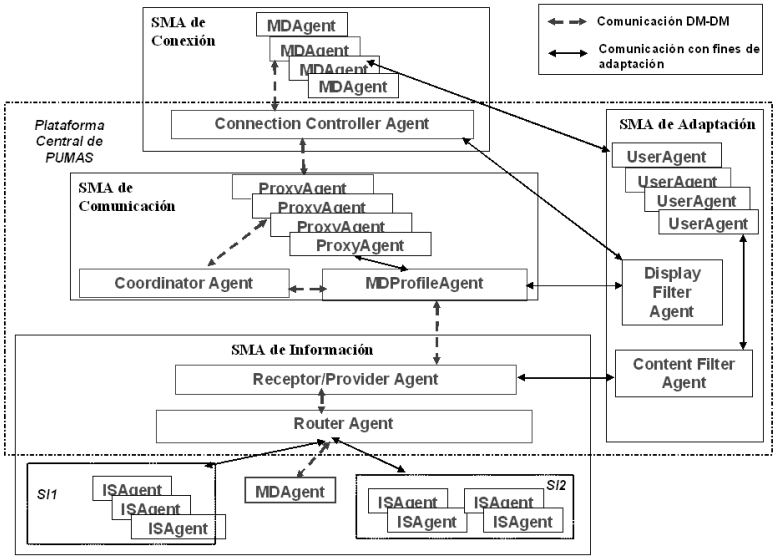

Fig. 1. Arquitectura de *PUMAS*

El *SMA* de *Conexión* provee los mecanismos que facilitan la conexión de diferentes tipos de *DM* al sistema (plataforma central de *PUMAS*).

El *SMA* de *Comunicación* garantiza una comunicación transparente entre los *DM* y el sistema. También aplica el *Filtro de Despliegue* (*Display Filter*) que consiste en mostrar al usuario a través de su *DM*, la información de acuerdo a las restricciones técnicas de su *DM*. Para hacer esto, este *SMA* cuenta con la ayuda de los agentes del *SMA* de *Adaptación*.

El *SMA* de *Información* recibe las consultas (*queries*) del usuario y las redirecciona hacia el *Sistema de Información* (*SI*) más *"adecuado"* (por ejemplo, el más cercano, o el *SI* que frecuentemente responde la consulta del usuario, o el más consultado), aplica el *Filtro de Contenido* (*Content Filter*), con la ayuda de los agentes del *SMA* de *Adaptación*, de acuerdo al perfil de

usuario en el sistema (es decir, sus preferencias, su historia) y retorna los resultados *"filtrados"* al *SMA* de *Comunicación*.

Los agentes del *SMA* de *Adaptación* se comunican con los agentes de los otros tres *SMA* con el fin de intercambiar información acerca del usuario (explícitamente extraída de los archivos *XML* o inferida de las reglas que administran el sistema), acerca de la conexión y comunicación, sobre las características del *DM*, etc. Los servicios y tareas desempeñados por el *SMA* de *Adaptación* consisten esencialmente en el manejo de archivos *XML* específicos que contienen información sobre el usuario y su *DM*. Los agentes del *SMA* de *Adaptación* manejan conocimiento (almacenado en *Bases de Conocimiento - BC*) que permite filtrar la información a los usuarios. Existe también conocimiento que es inferido a partir del análisis de la historia del usuario en el sistema, incluyendo sus últimas conexiones, consultas, preferencias, etc.

Una descripción más detallada de los componentes de *PUMAS* se encuentra en [3]. En las siguientes subsecciones se describe el conocimiento y los datos que maneja cada uno de los agentes de *PUMAS*.

3.1 El SMA de Conexión

Este *SMA* incluye uno o más *Mobile Device Agents* y un *Connection Controller Agent*.

El *Mobile Device Agent* (*MDA*) se ejecuta en el *DM* del usuario. Su conocimiento está compuesto de reglas generales de comportamiento y de características relacionadas con el tipo de *DM* usado (por ejemplo, *PDA*) y algunos roles específicos definidos de acuerdo con la aplicación (por ejemplo, este agente se usa para transmitir un archivo). Adicionalmente, un *MDA* debe conocer los protocolos de comunicación (es decir, conexión, tipo de red, restricciones, etc.) compartidos con el sistema. El *MDA* maneja archivos *XML* (*Device Profile XML*, localizado en el *DM* del usuario*,* ver Fig. 2) que describe las características del *DM* (utilizando *OWL*[3] con el fin de definir una ontología común para los agentes que manejan la información relacionada con las características de los *DM*). El *MDA* comparte esta información con el *DisplayFilterAgent* (agente que pertenece al *SMA* de *Adaptación*) a través del *Connection Controller Agent.* El *MDA* envía este archivo al *Connection Controller Agent* (este último agente se ejecuta en la plataforma central de *PUMAS*) e intercambia esta información con el *DisplayFilterAgent*. Este archivo contiene cierta información acerca de los requerimientos de hardware y de software, características de la aplicación que se ejecuta en esta sesión, el estado de la red, el tipo de archivos multimedia que el *DM* soporta, las condiciones para desconexión (es decir, finalización de sesión): sesión inactiva por más de *X* minutos y tipo de desconexión (voluntaria, involuntaria, automática).

Un *Mobile Device Agent* también maneja el *Current Session XML* que describe las características de la sesión del usuario: quién está conectado (*IDUsuario*), cuándo se inicio la sesión y cuál es el *DM* conectado. Este archivo se enviará al *UserAgent* (del *SMA* de *Adaptación*).

El *Connection Controller Agent* (*CCA*) se ejecuta en la plataforma central de *PUMAS* y obtiene la localización del usuario y el tipo de *DM* (por ejemplo, *PDA*) del archivo *User Location XML* (que contiene las características de la localización física y lógica del usuario) y del archivo *Device Profile XML*. Ambos archivos son provistos por los *Mobile Device Agents* y localmente manejados por el *Connection Controller Agent.*

El *Connection Controller Agent* sirve como intermediario entre los *SMA* de *Conexión* y el de *Comunicación.* Dicho agente también revisa las conexiones establecidas por los usuarios y el estado de los agentes (tales como, conectado, desconectado, eliminado, etc.), y asocia cada *Mobile Device Agent* a su correspondiente *Proxy Agent* (agente que se ejecuta en el *SMA* de *Comunicación,* ver sección siguiente).

---

[3] *OWL*: *Ontology Web Language* se construye a base de esquemas *RDF* y se caracteriza por la descripción de propiedades y clases (relaciones entre clases, cardinalidad, igualdad, definición de nuevos tipos de datos para las propiedades, características de las propiedades, y clases enumeradas). Ver *http://www.w3.org/2004/OWL/*

```
<?xml versión="1.0"?> <rdf:RDF
<owl:Ontology rdf:about=""/>
 <owl:Class rdf:ID="Profile-MIDP-PJava">
  <rdfs:subClassOf> <owl:Class rdf:ID="SoftwareRequirements"/> </rdfs:subClassOf>
 </owl:Class>
  <owl:Class rdf:ID="kVM">
  <rdfs:subClassOf> <owl:Class rdf:about="#SoftwareRequirements"/> </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Resolution">
  <rdfs:subClassOf> <owl:Class rdf:ID="Screen"/> </rdfs:subClassOf> </owl:Class>
 <owl:Class rdf:about="#Screen">
  <rdfs:subClassOf> <owl:Class rdf:about="#HardwareRequirements"/> </rdfs:subClassOf>
 </owl:Class>
  <owl:Class rdf:ID="Sound">
  <rdfs:subClassOf> <owl:Class rdf:ID="SupportedHyperMediaTypeFiles"/> </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="DisconnectionType">
  <rdfs:subClassOf> <owl:Class rdf:ID="DisconnectionStatus"/> </rdfs:subClassOf> </owl:Class>
 <owl:Class rdf:ID="InactivationBeforeDisconnection">
  <rdfs:subClassOf> <owl:Class rdf:about="#DisconnectionStatus"/> </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="ConnectionStatus">
  <rdfs:subClassOf rdf:resource="#DisconnectionStatus"/>  </owl:Class></rdf:RDF>
```

Fig. 2. Un fragmento del archivo *Device Profile XML*

Los archivos *XML* (*User Location, Current Session* y *Device Profile XML*) manejados por el *Mobile Device Agent* y el *Connection Controller Agent* son definidos usando las extensiones introducidas por Indulska *et al.* a *CC/PP* [5]. Estas extensiones incluyen algunas características del usuario como su localización (física y lógica), requerimientos de las aplicaciones disponibles (requerimientos de hardware, software, browser y *WAP*), características de la sesión (usuario, dispositivo, aplicación, etc.) y el perfil del usuario (requerimientos del usuario, preferencias, etc.). Usamos este tipo de archivos con el fin de manejar un estándar y una descripción común para los agentes que comparten información. Los agentes del *SMA* de *Conexión*, en especial los *Mobile Device Agents* que se ejecutan en *DM,* no manejan Bases de Datos *XML* para almacenar la información. El conocimiento de todos los agentes de *PUMAS* (propio, adquirido, etc.) es almacenado en *Bases de Conocimiento JESS*. Los archivos *XML* son traducidos por los agentes de la plataforma central del *PUMAS* en *"hechos"* (piezas de conocimiento) definidos en *JESS* (ver sección 5). Los *Mobile Device Agents* no hacen ningún tipo de traducción, sólo intercambian información con los agentes que se ejecutan en la plataforma central de *PUMAS*.

3.2 El SMA de Comunicación

Este *SMA* está compuesto de uno o más *Proxy Agents*, un *MDProfile Agent* y un *Coordinator Agent*. Estos agentes se ejecutan en la plataforma central de *PUMAS*.

Hay un *Proxy Agent* por conexión de un *Mobile Device Agent.* Dos usuarios diferentes pueden conectarse al sistema a través del mismo *DM* con lo cual se tendrían dos diferentes *Proxy Agents* y dos sesiones diferentes. La principal actividad del *Proxy Agent* es representar un *Mobile Device Agent* al interior del sistema. En este caso, hay dos agentes, un *Mobile Device Agent* que se ejecuta en el *DM* y un *Proxy Agent* que se ejecuta en la plataforma central de *PUMAS*. El *Proxy Agent* tiene las mismas propiedades y comportamiento que el *Mobile Device Agent* excepto aquellas concernientes a la conexión.

El *MDProfileAgent* debe revisar el perfil de usuario (de acuerdo a su *DM*). Adicionalmente, este agente junto con el *Coordinator Agent* define y revisa los mecanismos para enviar, por ejemplo, datos multimedia al usuario. Si los resultados de la consulta del usuario se traducen en varias imágenes, estos agentes definen el orden y el número de imágenes que serán mostradas a través de la pantalla del *DM* del usuario (de acuerdo a sus restricciones técnicas). El *MDProfileAgent* también comparte información con el *DisplayFilterAgent* (del *SMA* de *Adaptación*) sobre las características específicas del *DM* para la sesión del usuario.

El *Coordinator Agent* está en permanente comunicación con el *Connection Controller Agent* con el fin de verificar el estado de la conexión del agente que solicita la información. El *Coordinator Agent* conoce todos los agentes conectados en el sistema (utilizando un mecanismo de páginas amarillas) gracias a los archivos *XML* manejados por el *Mobile Device Agent* (a través del *Proxy Agent* que lo representa): sus conexiones, estado, servicios, localización, y características técnicas del *DM* del usuario. Si hay algún problema con el *Connection Controller Agent* (por ejemplo, si dicho agente falla o si hay más conexiones de las que puede manejar), el *Coordinator Agent* puede desempeñar el rol del *Connection Controller Agent* hasta que se solucionen los problemas. En ese momento, el *Connection Controller Agent* y el *Coordinator Agent* deben sincronizar la información sobre los agentes conectados y el estado de sus conexiones actuales.

3.3 El SMA de Información

El *SMA* de *Información* está compuesto de un *Receptor/Provider Agent*, un *Router Agent* y uno o más *ISAgents*.

El *Receptor/Provider Agent* que se ejecuta en la plataforma central de *PUMAS* tiene una visión general de todo el sistema. Este agente conoce tanto los agentes del *SMA* de *Comunicación* como los del de *Información*, sus servicios, sus localizaciones, sus perfiles, etc. El *Receptor/Provider Agent* recibe todas las consultas provenientes del *SMA* de *Comunicación* y las redirige al *Router Agent* que es el encargado de encontrar el *Sistema de Información "adecuado"* con el fin de ejecutar la consulta (ver sección 7). Una vez la consulta ha sido procesada por los *ISAgents* (agentes que se ejecutan en los *SIW*), el *Receptor/Provider Agent* verifica si los resultados de la consulta toman en cuenta el perfil del usuario (es decir, sus preferencias, su historia, sus intenciones, etc.), información provista por el *ContentFilterAgent* (del *SMA* de *Adaptación*). El *ContentFilterAgent* es el responsable de actualizar las preferencias del usuario almacenadas en su *Base de Conocimiento* y se comunica con el *UserAgent* (del *SMA* de *Adaptación*), que maneja la información del usuario para la sesión actual.

Con el fin de redireccionar la consulta hacia los *Sistemas de Información "adecuados"*, el *Router Agent* aplica una *estrategia* que depende de uno o más criterios: la localización del usuario, su historia en el sistema, actividades desarrolladas durante un periodo de tiempo, orientación de desplazamiento, preferencias, etc. La *estrategia* puede consistir en enviar la consulta a un *SIW* específico, o enviarla mediante *broadcast*, y/o dividir la consulta en subconsultas, cada una siendo enviada a uno o más *SIW*. El *Router Agent* también está a cargo de recopilar los resultados enviados por los *ISAgents* (que se ejecutan en los diferentes *SIW*) y de analizarlos (de acuerdo a los criterios definidos en las preferencias del usuario) para decidir qué parte de los resultados debe ser enviada al *Receptor/Provider Agent*. Con el propósito de enviar las consultas y analizar sus resultados, el *Router Agent* debe revisar las preferencias del usuario, información provista por el *ContentFilterAgent* a través del *Receptor/Provider Agent*.

Un *ISAgent* asociado con un *SIW* (y que se ejecuta en el mismo dispositivo del *SIW*) recibe del *Router Agent* las consultas del usuario y es el encargado de buscar la información al interior del *SIW*. Una vez el *ISAgent* obtiene los resultados de la consulta, los envía al *Router Agent*. Un *ISAgent* puede ejecutar una consulta o delegar esta tarea al componente adecuado del *SIW*. Esto depende de la naturaleza del *SIW*. Nuestra propuesta se dirige por una parte, a *SIW* complejos y posiblemente distribuidos, localizados en servidor(es) y por otra parte, a *SIW* simples consistentes en algunos archivos almacenados en un *DM*. En este último caso, un *ISAgent* es suficiente para asegurar el correcto funcionamiento del *SMA* de *Información*. Vale la pena anotar, que en este caso lo que llamamos un *"ISAgent"* es verdaderamente un *Mobile Device Agent* de un *DM* que puede jugar el rol de un *ISAgent* ya que cuenta con el conocimiento requerido para ejecutar la consulta, buscando información en los archivos almacenados en el *DM*. En un *SIW* complejo, el *ISAgent* puede colaborar con otro *ISAgent* (si el *SIW* ha sido desarrollado siguiendo el paradigma de *SMA*) o con otro componente del *SIW* para ejecutar la consulta. En el caso de un *SIW* no desarrollado como un *SMA*, nuestra propuesta sólo requiere que haya un *ISAgent* que se ejecute en dicho *SIW*,

que pueda buscar información en su interior y que garantice la comunicación entre *PUMAS* y el *SIW*. Debido a que la localización de un *SIW* puede cambiar (especialmente si éste se ejecuta en un *DM*), el *Router Agent* puede ser informado sobre dicha localización a través del *ISAgent* que se ejecuta en el *SIW*.

3.4 El SMA de Adaptación

Las capacidades de adaptación de *PUMAS* se basan en un proceso de filtro compuesto de dos etapas cuyo objetivo es brindarle al usuario información adaptada a sus necesidades y características así como a su *DM*. (es decir, *"la información adecuada, en el lugar y momento adecuados"*). Primero, *el Filtro de Contenido* (*Content Filter*) permite seleccionar la información más relevante de acuerdo al perfil de usuario definido en el sistema. Segundo, el *Filtro de Despliegue* (*Display Filter*) se aplica a los resultados del primer filtro y toma en cuenta las características y restricciones técnicas del *DM* del usuario.

El *SMA* de *Adaptación* está conformado por uno o más *UserAgents*, un *DisplayFilterAgent* y un *ContentFilterAgent*. Estos agentes se ejecutan en la plataforma central de *PUMAS*.

Cada *UserAgent* extrae información de los archivos *XML* enviados por el *Mobile Device Agent* (del *SMA* de *Conexión*), que contienen las características del usuario (usuario *ID*, localización, etc.) y sus preferencias (por ejemplo, *"el usuario desea sólo información en video"*). El archivo *User Profile XML* (que es enviado por el *Mobile Device Agent* al *UserAgent*) contiene dichas preferencias. Sólo hay un *UserAgent* por usuario en un momento determinado (incluso si el usuario tiene abierta dos sesiones en uno o diferentes *DM*). Debido a que un usuario puede acceder el sistema a través de diferentes *DM,* el *UserAgent* se comunica con los *Mobile Device Agents* y los *Proxy Agents* (estos últimos del *SMA* de *Comunicación*) para analizar y centralizar todas las características del mismo usuario. El *UserAgent* se comunica con el *ContentFilterAgent* para enviar el archivo *User Profile XML*. Cuando el *ContentFilterAgent* recibe este archivo, almacena esta información en su *Base de Conocimiento* (es decir, este agente almacena las preferencias de usuario). Cuando el *Receptor/Provider Agent* (del *SMA* de *Información*) solicita al *ContentFilterAgent* las preferencias del usuario, el *ContentFilterAgent* le envía el último archivo *User Profile XML* enviado por el *UserAgent*. Si el *UserAgent* no le ha enviado este archivo (por ejemplo, no hay preferencias específicas para la sesión actual), el *ContentFilterAgent* toma en cuenta para este usuario las preferencias de sus sesiones anteriores.

El *DisplayFilterAgent* maneja una *Base de Conocimiento* que contiene información general acerca de las características de los diferentes tipos de *DM* (por ejemplo, los formatos de archivo que soporta) y el conocimiento adquirido de conexiones anteriores (por ejemplo, problemas y capacidades de redes de acuerdo a la transmisión de datos). El *Connection Controller Agent* (del *SMA* de *Conexión*) se comunica con el *DisplayFilterAgent* con el fin de solicitarle información sobre características y/o problemas de conexión (tales como, capacidades de ancho de banda, velocidad de transmisión, etc.). El *MDProfile Agent* (del *SMA* de *Comunicación*) también se comunica con el *DisplayFilterAgent* para solicitarle información sobre las capacidades y restricciones de un *DM*.

El *ContentFilterAgent* maneja una *Base de Conocimiento* que contiene las preferencias, intenciones y características de los usuarios. El *ContentFilterAgent* se comunica con el *UserAgent,* solicitándole información sobre las preferencias de un usuario para una sesión específica (por ejemplo, la sesión actual). Si el *UserAgent* tiene preferencias específicas para una sesión, el *ContentFilterAgent* se las comunica al *Receptor/Provider Agent* (del *SMA* de *Información*). De otra manera, el *ContentFilterAgent* busca las preferencias y características del usuario (provenientes de sesiones anteriores) en su *Base de Conocimiento*. Las características del usuario en el sistema son solicitadas por el *Receptor/Provider Agent* que las adiciona a las consultas del usuario. Este agente verifica si los resultados son adaptados de acuerdo a esta información.

4 Escenarios de PUMAS

En esta sección, presentamos algunos escenarios que muestran cómo los agentes de *PUMAS* interactúan cuando un usuario se conecta, envía una consulta y cuando el sistema retorna los resultados de dicha consulta. En nuestra propuesta, las interacciones entre agentes se basan en el intercambio de mensajes de acuerdo con los *Actos de Comunicación* (*Communication Acts*) presentados por Odell *et al.* en [8] (tales como, confirm, inform, propose, request, propagate).

4.1 Escenario de conexión

Cuando un usuario se quiere conectar al sistema a través de su *DM*, el *Mobile Device Agent* que se ejecuta en el *DM* del usuario envía un mensaje de *"propose"* (proposición de conexión) al *Connection Controller Agent*. Si no hay un *Proxy Agent* (*PA*) que represente este *Mobile Device Agent*, el *Connection Controller Agent* crea uno y le envía un mensaje de *"subscribe"* al *Coordinator Agent* con el fin de suscribir este *Proxy Agent* al sistema. El *Coordinator Agent* informa al *MDProfile Agent* de esta suscripción. El *Connection Controller Agent* también envía al *Mobile Device Agent* un mensaje de *"confirms"* cuando el proceso de suscripción ha terminado (ver Fig. 3). Cuando el *Mobile Device Agent* recibe el mensaje de confirmación, se crea un *UserAgent* (*UA*) en la plataforma central de *PUMAS* con el fin de manejar el perfil de usuario. Este perfil se define en el archivo *Current Session XML* y es enviado por el *Mobile Device Agent* al *UserAgent.*

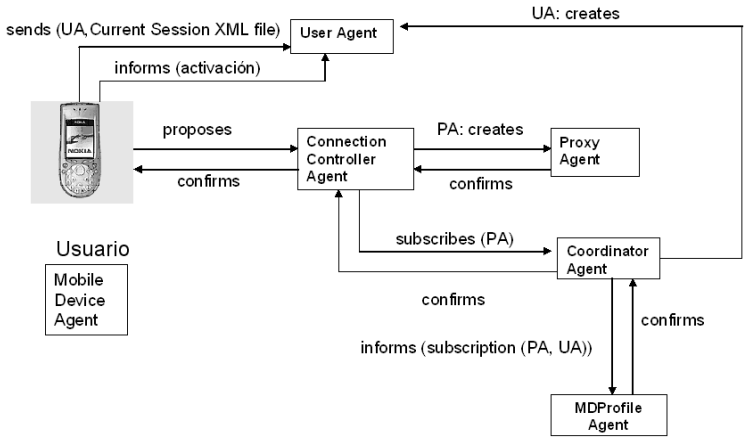

Fig. 3. Escenario de conexión en PUMAS

4.2 Escenario de envío de una consulta

Cuando un usuario envía una consulta *C* (ver Fig. 4), el *Mobile Device Agent* la envía (a través de un mensaje) al *Connection Controller Agent.* Si la consulta *C* depende de la localización y del tiempo de conexión (dependencia definida en las preferencias del usuario, ver sección 5.2), el *Connection Controller Agent* introduce esta información a la consulta *C,* así como las características de conexión del *DM* (esto último conocido a través del *DisplayFilterAgent*). Esta nueva consulta *C'* (en Fig. 4, *C'=C + localización*) es enviada al *Proxy Agent*. *C'* pasa por el *Coordinator Agent* y luego por el *MDProfile Agent*. Este último adiciona a la consulta *C'* algunas características relacionadas con el *DM;* dichas características son provistas por el *DisplayFilterAgent* que las ha aprendido de consultas anteriores o que las ha recuperado de su *Base de Conocimiento*. La nueva consulta *C"* (en Fig. 4, *C"= C' + características del DM*) es enviada por el *MDProfile Agent* al *Receptor/Provider Agent.* Este último agente adiciona a *C"* las características específicas de usuario que el *ContentFilterAgent* le ha dado (En Fig. 4, *C'''=C" +*

Preferencias del usuario). El *Receptor/Provider Agent* envía la *C'''* al *Router Agent* que decide (de acuerdo a la consulta, el sistema de reglas y los hechos de su *Base de Conocimiento*) qué *ISAgents* (agentes que se ejecutan en los *SIW*) son capaces de responderla. El *Router Agent* puede enviar la consulta a un *ISAgent* específico o a varios (por ejemplo, esperando el primero en responder o diferentes respuestas) o, puede dividir la consulta en subconsultas que pueden ser enviadas a uno o más *ISAgents*. El escenario de la Fig. 4 muestra cómo la consulta *C'''* se divide en $C'''_{-1.1}$, $C'''_{-1.2}$, $C'''_{-1.3}$ y $C'''_{-1.4}$. Dichas subconsultas son enviadas a los *ISAgents* que se ejecutan en un servidor y en diferentes *DM*.

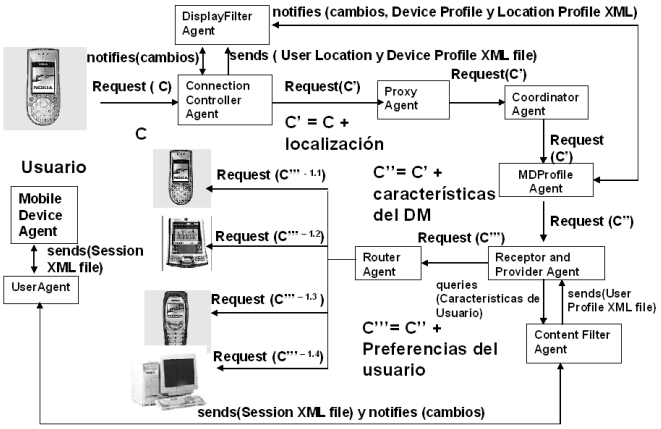

Fig. 4. Escenario de envío de una consulta.

Cuando un usuario *U1* tiene una consulta para otro usuario (*U2*), ambos equipados con *DM,* la consulta se propaga desde el *Mobile Device Agent* que se ejecuta en el *DM* de *U1* hasta el *Router Agent,* agente que la redirige al *Mobile Device Agent* que se ejecuta en el *DM* de *U2*. El *Mobile Device Agent* de *U2* cambia de rol y llega a ser un *ISAgent,* es decir, el agente a cargo de responder la consulta. Este cambio de rol es posible porque un *Mobile Device Agent* tiene el conocimiento para manejar la información almacenada en el *DM* en el cual se ejecuta y además, cuenta con la capacidad de responder las consultas.

4.3 Escenario de recepción de los resultados de una consulta

Cuando el *Router Agent* recibe de los *ISAgents* todos los resultados de la consulta (en la Fig. 5, el *Router Agent* recibe los resultados parciales – $R^{1.1}$, $R^{1.2}$, $R^{1.3}$ y $R^{1.4}$ – enviados por los *ISAgents* que se ejecutan en un servidor y diferentes *DM*), el *Router Agent* los analiza antes de enviar mensajes de *"confirms"* (está de acuerdo con los resultados) o *"disconfirms"* (no está de acuerdo con los resultados) o *"not understand"* (no entendió los resultados) al *Receptor/Provider Agent.* Este mensaje incluye los resultados de la consulta (*R*). El *Receptor/Provider Agent* revisa si los resultados satisfacen las características específicas del usuario (en la Fig. 5, *R'* es el resultado de aplicar el *Filtro de Contenido* a *R* de acuerdo a las preferencias del usuario) y las redirige al *MDProfile Agent*. Este agente verifica si los resultados pueden ser desplegados de acuerdo a las características del *DM* y lleva a cabo el primer paso del *Filtro de Despliegue* (*Display Filter*). En la Fig. 5 podemos observar que *R"* es el resultado de filtrar *R'* de acuerdo a las características del *DM*. Luego, *R"* se transmite desde el *Coordinator Agent* hasta el *Proxy Agent.* Este último lo envía al *Connection Controller Agent* que ejecuta el último paso del *Filtro de Despliegue* de acuerdo a las características técnicas y de conexión del *DM* (si el usuario está aún conectado y ha cambiado de localización, si ha ocurrido un *timeout*, etc.). Gracias a los filtros de *Contenido* y de *Despliegue*, los resultados de la consulta recibidos por el *Mobile Device Agent* y desplegados en el *DM* del usuario corresponden a la información más relevante para el usuario.

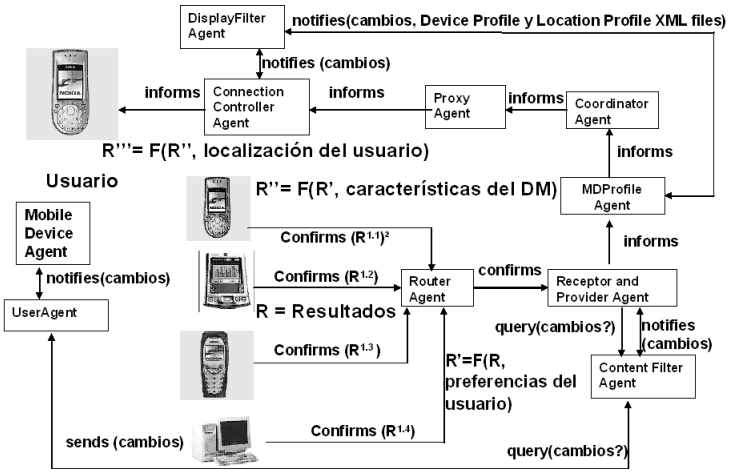

Fig. 5. Escenario de recepción de los resultados de una consulta

Vale la pena anotar que el escenario descrito arriba incluye varias revisiones que podrían parecer inútiles ya que en el escenario de envío de una consulta se tienen en cuenta tanto las características del usuario como las de su *DM*. Sin embargo, esta información extra adicionada durante el proceso de envío de la consulta podría no ser válida en el momento en el que se entregan los resultados de dicha consulta al usuario. Las características del usuario (tales como, cambios de localización, de preferencias) y los medios de comunicación y conexión (tales como, variación de ancho de banda, uso de diferentes *DM*) podrían haber cambiado y tener un impacto sobre los resultados esperados. Los controles que se presentan en el escenario de recepción de resultados de una consulta se orientan a eliminar o reducir información que resulte ser irrelevante ante los cambios ocurridos.

5 Manejo del conocimiento en PUMAS

En esta sección describimos el conocimiento que es manejado por los agentes de los *SMA* de *Información* y de *Adaptación* de *PUMAS* para llevar a cabo su labor de adaptación de la información. Este conocimiento es almacenado en *Bases de Conocimiento* (*BC*) como *"piezas de conocimiento"*. Llamamos estas piezas *"hechos"* y las definimos usando *JESS*[4]. Declaramos estos hechos como instancias de *Templates JESS* con el fin de representar las preferencias del usuario, las características del *DM*, los *Sistemas de Información*, etc., tal y como se describe en las siguientes subsecciones.

5.1 Conocimiento del SMA de Información

El *Router Agent* (*RA*) almacena en su *Base de Conocimiento,* un hecho por cada *Sistema de Información* (*SI*). El *RA* utiliza estos hechos para redireccionar las consultas del usuario. Un hecho que representa un *SI* describe sus características como su nombre, la información que maneja, el tipo de dispositivo donde se ejecuta (por ejemplo, servidor, *PC*, *DM*) y el *ISAgent* (*ISA*) asociado con el *SI* (es decir, el agente que puede consultar la información dentro del *SI* y consecuentemente responde dichas consultas). Por ejemplo, el siguiente template *JESS* define un *SI*:

[4] *JESS* es un motor de inferencia y un ambiente de scripts para construir aplicaciones *JAVA* que tienen la capacidad de "razonar" usando conocimiento suministrado como reglas declarativas. Ver *http://herzberg.ca.sandia.gov/jess/*

(*deftemplate*[5] *SI* (*slot nombre*) (*slot IDAgente*) (*slot dispositivo*) (*multislot info_items*)) ; *hecho* (*1*)[6]

El siguiente hecho (instancia del template definido anteriormente) representa el *Sistema de Información* del Laboratorio Clínico de un hospital. El *Sistema de Información* se llama *SILabClínico* y se ejecuta en un *servidor*. El *ISAgent* que se ejecuta en este *Sistema de Información* es el *LabClínicoISA*. El *SILabClínico* contiene información (una lista llamada *info_items*) relacionada con los resultados de los exámenes (análisis) de los pacientes, los reactivos y las indicaciones especiales para dichos exámenes (por ejemplo, ayuno):

(*assert*[7] (*SI* (*nombre SILabClínico*) (*IDAgente LabClínicoISA*) (*dispositivo servidor*) (*info_items "resultados de exámenes de pacientes" "reactivos" "indicaciones"*)))

5.2 Conocimiento del SMA de Adaptación

Consideramos que las consultas pueden depender de uno o más criterios: la localización del usuario, su historia en el sistema, sus actividades desarrolladas durante un periodo de tiempo, sus preferencias, etc. Tales Criterios de Dependencia (*CriterioDependencia*) son definidos como:

(*deftemplate CriterioDependencia* (*slot usuarioID*)(*multislot criterios*)(*multislot atributos*)) ; *hecho* (*2*)

El siguiente ejemplo de *CriterioDependencia* expresa que todas las consultas del *Doctor John Smith* dependen de su localización, especialmente si él está en su trabajo (*Hospital del Norte*):

(*assert* (*CriterioDependencia* (*usuarioID "Doctor John Smith"*)(*criterios localización*)(*atributos "Hospital del Norte"*)))

El *DisplayFilterAgent* maneja una *Base de Conocimiento* que contiene información general sobre las características de los diferentes tipos de *DM*. Cada *CaracterísticaDM* es definida como un hecho y se representa como sigue:

(*deftemplate CaracterísticaDM* (*slot tipoDM*) (*multislot característica*)) ; *hecho* (*3*)

Donde cada *característica* es representada como un hecho de la siguiente manera:

(*deftemplate característica* (*slot tipo*)(*multislot condiciones*)) ; *hecho* (*4*)

Las *condiciones* de una *característica* son aquellas que ésta necesita para ser satisfecha (por ejemplo, el tipo de red necesitada para soportar algún tipo de dato). A continuación damos un ejemplo de una *característica* que corresponde al formato de archivos soportados por una *Palm Tungsten C* en diferentes tipos de red. Suponemos que este *DM* puede soportar video y varias imágenes usando una red *Wi-Fi*:

(*deffacts CaracterísticaDM* (*tipoDM "Palm Tungsten C"*)
(*característica* (*tipo video_soportado*) (*condiciones "red Wi-Fi"*))
(*característica* (*tipo varias_imágenes*) (*condiciones "red Wi-Fi"*))) ;

El *ContentFilterAgent* maneja una *Base de Conocimiento* que contiene información acerca de las preferencias del usuario. Estas preferencias son representadas como hechos definidos de la siguiente manera:

(*deftemplate Preferencia_Usuario* (*slot IDUsuario*)
(*slot info_requerida*) (*multislot info_complementaria*)
(*multislot acciones_a_ejecutar*)
(*slot problema*)(*multislot acciones_para_recuperar*))) ; *hecho* (*5*)

El hecho que representa una *Preferencia_Usuario* está compuesto del identificador del usuario propietario de esta preferencia (*IDUsuario*), la información requerida (*info_requerida*) y la información complementaria (*info_complementaria*). El *ContentFilterAgent* adiciona esta *info_complementaria* a la *Preferencia_Usuario*. El *ContentFilterAgent* analiza las consultas

[5] Definimos nuestras piezas de conocimiento usando la sintaxis de *hechos no ordenados* (*unordered facts*) de *JESS*. Declaramos cada hecho no ordenado mediante la primitiva *"deftemplate"*.
[6] Enumeramos los hechos que vamos a utilizar en la sección 7.
[7] Para definir una instancia de un hecho no ordenado en *JESS* y para luego insertarla en la *Base de Conocimiento JESS* usamos la primitiva *"assert"*.

efectuadas en sesiones anteriores (información frecuentemente solicitada de manera simultánea con la *info_requerida*). Este hecho también se compone de la información acerca de qué y cómo le gustaría al usuario que el sistema despliegue los resultados en el *DM* (*acciones_a_ejecutar*) y en caso de problemas, qué acciones el sistema debe ejecutar (*acciones_para_recuperar*). Para esto, cada *acción* es definida como un hecho y representada como sigue:

(*deftemplate acción* (*slot nombre*)(*multislot atributo*)) *; hecho* (*6*)

En esta definición, *nombre* se refiere a una acción escogida de una lista definida (tales como, mostrar, salvar, transferir archivo, cancelar). Cada acción tiene una lista de *atributos*. Por ejemplo, la acción *"mostrar"* que tiene como *atributos* el *orden,* el *formato* y el *tipo* de *archivo* que va a ser mostrado, se define como un hecho de la siguiente manera:

(*assert* (*acción* (*nombre mostrar*)(*atributo "orden" "formato" "tipo_archivo"*)))

Debido a que un *atributo* puede ser complejo, lo definimos como un hecho de la siguiente manera:

(*deftemplate atributo* (*slot nombre*)(*multislot lista*)) *; hecho* (*7*)

Un ejemplo de *atributo* que define el *orden* en el cual la información es desplegada en el *DM* es:

(*assert* (*atributo* (*nombre orden*)(*lista "dieta del paciente" "análisis/exámenes sanguíneos" "medicamentos prescritos"*)))

Podemos definir un *problema* como algo inesperado o no deseado que ocurre cuando una acción es ejecutada (por ejemplo, demora en la transmisión de datos en una localización determinada o bajo ciertas condiciones de red), o, que es la consecuencia de la ejecución fallida de una acción (por ejemplo, imposible mostrar una gran imagen a través de determinado *DM*). Cada *problema* es definido como un hecho y representado como sigue:

(*deftemplate problema* (*slot nombre*)(*slot tipo*)(*multislot causas*)) *; hecho* (*8*)

Donde *nombre* corresponde a una descripción del problema, el *tipo* puede ser escogido de una lista definida (tales como, *incompatibilidad, Sistema de Información no disponible*), y las *causas* corresponden a una lista de posibles causas del problema (por ejemplo, el *DM* no puede soportar un formato de archivo específico por restricciones técnicas, problemas de red). Por ejemplo, el siguiente hecho define el problema de un *DM* que no puede soportar algún tipo de información multimedia (por ejemplo, un video). El problema definido abajo es considerado como una *incompatibilidad* causada por la reducida *memoria_DM* y por el reducido *tamaño_pantalla*:

(*assert* (*problema* (*nombre videonosoportado*) (*tipo incompatibilidad*)(*causas "memoria_DM" "tamaño_pantalla"*)))

6 Ejemplo

En esta sección, ilustramos el proceso ejecutado por los agentes de *PUMAS* mediante un *SIW* de un hospital. Para este ejemplo supongamos que doctores equipados con *DM* (por ejemplo, *PDA*) acceden al *SIW* de un hospital. Dicho *SIW* puede estar distribuido entre diferentes *DM* y/o uno o más *SIW* (ver Fig. 6). Los doctores pueden también recibir información que concierne a sus pacientes de acuerdo a su localización, preferencias, características técnicas de sus *DM* y consideraciones acerca del momento en el que se conectan. Por ejemplo, al visitar un paciente, los doctores con *DM* pueden consultar información sobre la historia clínica del paciente, sus análisis médicos, los medicamentos, etc. Mediante la localización del paciente (cuarto, piso, cama, etc.) y la fecha actual, el doctor puede identificar el paciente y obtener su información. Para esto, la aplicación que se ejecuta en su *DM* debe consultar los diferentes *Sistemas de Información* del hospital – farmacia, pacientes, doctores, etc. Los doctores podrían también comunicarse con sus colegas a través de sus *DM* mediante consultas específicas dirigidas a ellos (por ejemplo, consultas que podría responder sólo el médico especialista que ha examinado recientemente al paciente).

Cuando un doctor ingresa la información concerniente a la localización del paciente y la aplicación toma la fecha del sistema (información sobre el tiempo de conexión), el *Mobile Device*

Agent que se ejecuta en su *DM* envía la consulta que es propagada a través de la plataforma central de *PUMAS*: primero se transmite hacia el *Connection Controller Agent*, luego a los agentes del *SMA* de *Comunicación* (*Proxy Agent, Coordinator Agent* y *MDProfile Agent*). El *MDProfile Agent* adiciona a la consulta, la información del *DM* (por ejemplo, si cierta clase de *DM* no soporta archivos gráficos, sólo texto, entonces cuando el doctor solicite los exámenes clínicos de un paciente, sólo obtendrá los resultados en formato texto). Así, si el doctor se conectó a través de una *Palm Tungsten C,* el *MDProfile Agent* puede solicitar al *DisplayFilterAgent* información acerca de este *DM.* El *MDProfile Agent* podría recibir del *DisplayFilterAgent* hechos como el siguiente:

(*deffacts Característica* (*DMtipo "Palm Tungsten C"*)
(*característica* (*tipo video_no_soportado*)(*condiciones red_Wi-Fi*))
(*característica* (*tipo varias_imágenes*)(*condiciones red_Wi-Fi*)
(*característica* (*tipo texto*)(*condiciones red_Wi-Fi red_Clasica Bluetooth*))) *;*

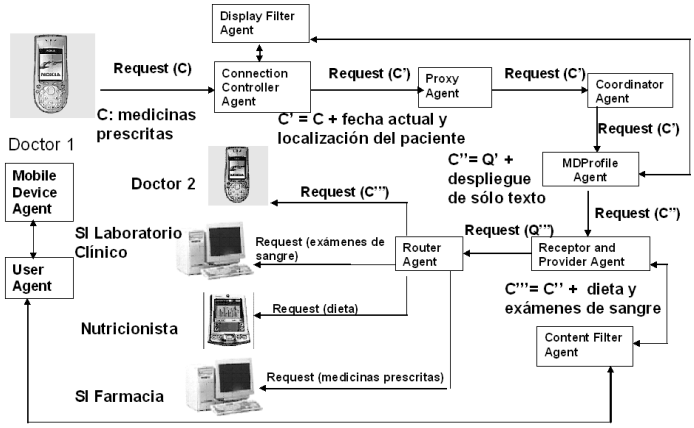

Fig. 6. Envío de una consulta en el *SIW* de un hospital

Posteriormente, el *MDProfile Agent* envía la consulta al *Receptor/Provider Agent* el cual le adiciona las preferencias que el doctor ha expresado previamente. Las preferencias se encuentran registradas en el archivo *User Profile XML* (ver sección 3.4) y son traducidas en hechos por el *UserAgent* y el *ContentFilterAgent* (que se ejecutan en la plataforma central de *PUMAS*). El siguiente ejemplo corresponde a una preferencia de un doctor: *"al solicitar los resultados de los análisis de sangre de un paciente, el sistema también debe proveer la dieta y los medicamentos prescritos a dicho paciente; se prefiere los resultados de una manera gráfica pero si el DM no soporta este formato, los resultados se recibirán en formato texto"*. El *UserAgent* traduce esta preferencia como el siguiente hecho:

(*deffacts Usuario_Preferencia* (*IDusuario "Doctor Smith"*)(*info_requerida "análisis de sangre"*)
(*info_complementaria "dieta del paciente" "medicamentos prescritos"*) (*acción mostrar*)
(*atributos* (*nombre orden*)(*lista "análisis del paciente" "dieta del paciente" "medicamentos prescritos"*))
(*atributos* (*nombre formato_gráfico*) (*lista "JPEG"*)) (*problema* (*nombre DatosHiperMediaNoSuportadosPorDM*)
(*tipo incompatibilidad*) (*causas SoloTextoSuportado*))(*atributos* (*nombre orden*) (*lista "análisis del paciente" "dieta del paciente" "medicamentos prescritos"*))(*atributos* (*nombre formato_texto*) (*lista "XML" "txt"*)))

El *UserAgent* transfiere esta información al *ContentFilterAgent* que almacena este hecho y lo envía al *Receptor/Provider Agent.* Este último agente adiciona esta preferencia a la consulta y la envía al *Router Agent*. El *Router Agent* recibe la consulta completa y con la información que éste tiene sobre los *Sistemas de Información*, puede dividirla en subconsultas, redirigiendo cada una de ellas hacia el *Sistema de Información* apropiado (ver sección 7). Los siguientes hechos son utilizados en este ejemplo por el *Router Agent* para redirigir las subconsultas a los *ISAgents* de los *Sistemas de Información* del hospital:

(*assert* (*SI* (*nombre SILabClínico*) (*IDAgente LabClínicoISA*) (*dispositivo servidor*)
(*info_items "resultados de exámenes de pacientes" "reactivos" "indicaciones"*)))

(*assert* (*SI* (*nombre SIDieta Paciente*) (*IDAgente ISADieta*) (*dispositivo DM*)
(*info_items "dieta de paciente" "citas del nutricionista"*)))

El *Router Agent* redirige la consulta al *ISAgent* localizado en cada uno de los *Sistemas de Información* que manejan la información acerca de los pacientes en el hospital. Todas las consultas siguen el mismo recorrido desde el *Mobile Device Agent* hasta el *Router Agent*. Si el doctor quiere conocer los últimos medicamentos prescritos a este paciente, el *Router Agent* redirige la consulta al *ISAgent* localizado en el *Sistema de Información* de la Farmacia. Si la consulta concierne a otro doctor, el *Router Agent* redirige la consulta al *ISAgent* localizado en el *DM* de dicho doctor. Un doctor también puede solicitar información de un paciente específico a varios de sus colegas. En este caso, el *Router Agent* podría enviar la consulta a manera de *broadcast* o podría dividirla de acuerdo al colega receptor (por ejemplo, consultas relacionadas con el corazón al cardiólogo) o a los criterios definidos en el archivo *User Profile XML* (por ejemplo, si el criterio de dependencia de la consulta es la *localización*, las consultas deben ser solamente redirigidas a los doctores que se encuentran en la misma *localización* o una cercana del emisor). La información recuperada es organizada por el *Router Agent* (por ejemplo, los medicamentos prescritos recientemente, las respuestas de los colegas acerca de un paciente) y es regresada al doctor quien envió la consulta siguiendo el recorrido inverso. Los diferentes agentes revisan los resultados porque, por ejemplo, el doctor puede haberse desconectado del sistema (debido a problemas de red), y recuperado su sesión a través de una nueva conexión cuyas características son diferentes de las anteriores: podría ser que el doctor en este momento consulta el sistema a través de otro *DM* que sí soporta grandes imágenes (lo que conlleva a que una de las preferencias del doctor pueda ser ahora satisfecha).

A través de este ejemplo, podemos observar el comportamiento de *PUMAS* como un sistema *P2P híbrido*. El núcleo de *PUMAS* centraliza las consultas y: *i)* está a cargo del proceso de obtener la información más relevante, *ii)* está a cargo de aplicar los filtros de *Contenido* y de *Despliegue* para adaptar las respuestas y *iii)* traduce los archivos *XML* en *"hechos"*, piezas fundamentales del conocimiento de los agentes. Las principales características de un sistema *P2P* que poseen los agentes de *PUMAS* se ilustran en el hecho de que en primer lugar, los agentes tienen la autonomía de conectarse y desconectarse del sistema. En segundo lugar, un *DM* puede comunicarse con un *Sistema de Información* específico (localizado en un servidor o en un *DM*) pasando esta información como un parámetro de la consulta; el *Router Agent* transmite la consulta a este *Sistema de Información* específico que ejemplifica una comunicación agente-agente (por ejemplo, cuando los doctores intercambian información acerca de un paciente usando sus *DM*).

En la siguiente sección, describimos el proceso de *enrutamiento de consultas* (*Query Routing*) que es especialmente llevado a cabo por el *Router Agent,* agente que aprovecha y utiliza el conocimiento que hemos descrito en la sección 5.

7 Query Routing en PUMAS

El proceso de *Query Routing* en *PUMAS* es llevado a cabo por el *Router Agent* (*RA*)*,* agente que recibe las consultas del usuario así como sus características y las de su *DM*. Con el fin de redirigir dichas consultas hacia los *Sistemas de Información "adecuados"*, la estrategia escogida por el *RA* depende de varios criterios: localización del usuario, similitud de sus colegas, tiempo de conexión, preferencias, etc. La estrategia puede estar orientada hacia el envío de la consulta a un *SIW* específico, o al envío de dicha consulta a manera de *broadcast*, o a dividir la consulta en subconsultas, cada una de ellas siendo enviada a uno o más *ISAgents* (agentes del *SMA* de *Información* que se ejecutan en el *SIW* y que buscan la información solicitada). El *RA* también es el encargado de recopilar los resultados provenientes de los *ISAgents* y de analizarlos (de acuerdo a los criterios definidos para las consultas, ver sección 5.2) con el fin de decidir si todo el conjunto o sólo una parte de ellos será enviado al *Receptor/Provider Agent* (que pertenece al *SMA* de *Información*).

El *Receptor/Provider Agent* conoce los agentes tanto del *SMA* de *Comunicación* como del de *Información*, sus servicios, su localización, sus perfiles, etc. Dicho agente recibe todas las consultas que son transmitidas desde el *SMA* de *Comunicación* y las redirecciona al *RA* que se encarga de encontrar el *Sistema de Información* que ejecuta la consulta. Una vez la consulta ha sido procesada por los *ISAgents*, el *Receptor/Provider Agent* verifica si los resultados toman en cuenta el perfil del usuario (información obtenida a través del *ContentFilterAgent*).

En *PUMAS*, el proceso de *Query Routing* está compuesto de tres actividades (basadas en el trabajo de Xu *et al.* [11]) que son descritas en las siguientes subsecciones. Explicamos cada actividad e ilustramos todo el proceso de *Query Routing* mediante el ejemplo del *SIW* del hospital.

7.1 Análisis de la consulta

Esta actividad está relacionada con la posible división de la consulta. El *RA* analiza la complejidad de la consulta. Una consulta es *simple* si sólo un agente la responde o, *compleja* si varios agentes son requeridos para responderla. Este análisis se basa en los hechos que el *RA* tiene almacenados en su *Base de Conocimiento* acerca de los *Sistemas de Información*, los criterios de dependencia de la consulta (por ejemplo, localización, similitud de los otros agentes), el conocimiento de los agentes que reciben las consultas (si la consulta está dirigida a un agente específico y conocido), etc. Después de este análisis, el *RA* decide si debe o no dividir la consulta en subconsultas.

Para la consulta que el *Doctor Smith* hace sobre un paciente (*"al solicitar los resultados de los análisis de sangre de un paciente, el sistema también debe proveer la dieta y los medicamentos prescritos a dicho paciente"*), el *RA* la debe dividir en tres subconsultas ("análisis sanguíneos", "dieta" y "medicamentos prescritos") ya que no existe un sólo *SIW* que pueda responderla (por consiguiente, se considera como una consulta *compleja*). Dicha división se basa en el conocimiento almacenado en su *Base de Conocimiento* acerca de los *SIW* (ver sección 5.1).

7.2 Selección de las fuentes de información

Una consulta puede ser dirigida a un agente específico o a un grupo de agentes; si los agentes que reciben la consulta son conocidos (la consulta está dirigida a receptores específicos), la selección de las fuentes es simple (las fuentes de información son los agentes específicos). De lo contrario, el *RA* selecciona las fuentes de información y crea la *"red de vecinos"*[8] (basados en las ideas de Yang *et al.* [12]). Consideramos diferentes casos para conformar la *red de vecinos* en la cual cada nodo es una fuente de información (*agente*):

Primero, podría haber varios agentes que pueden responder la misma consulta (por ejemplo, la dieta de un paciente podría encontrarse en el *SIW* de los Nutricionistas del hospital o podría estar almacenada en el *DM* de un nutricionista específico). La forma más simple de conformar la red es agrupar todos estos agentes. Este agrupamiento es útil, por ejemplo, cuando el *RA* no tiene información acerca de las fuentes o cuando es la primera vez que el *RA* trabaja con los potenciales vecinos. Con el fin de reducir comunicaciones que puedan resultar redundantes, innecesarias o inútiles y seleccionar los vecinos más relevantes, el *RA* aplica los criterios de dependencia para dicha consulta. Por ejemplo, si el criterio de dependencia es la *localización*, la red de vecinos será conformada por los *vecinos más cercanos*; si la consulta del usuario depende de *consultas anteriores*, el *RA* debe redireccionarla a los *vecinos más confiables* (aquellos que siempre han respondido de una manera adecuada, o más rápidamente, etc.); si el criterio de dependencia es la *similitud*, la red será conformada por los vecinos con un *perfil similar*, o que desempeñan *tareas similares*, entre otros. Si no hay criterio de dependencia establecido, el *RA* analiza el nivel de

[8] En nuestra proposición, un agente es *vecino* de otro si satisface un conjunto de características (criterios definidos en las preferencias de usuario de la aplicación). Por ejemplo, la proximidad en la localización, las mismas actividades, el mismo rol, conocimiento similar, colegas que trabajan en un mismo grupo, etc. Las características no se restringen al criterio de proximidad en la localización.

confianza de estos vecinos. Dicho agente asocia a cada vecino un nivel de confianza de acuerdo a anteriores interacciones con sus vecinos (basados en el trabajo de Agostini *et al.* [1]).

Segundo, una consulta podría ser respondida por sólo un agente y éste es conocido (por ejemplo, se sabe que los medicamentos prescritos a un paciente se encuentran registrados en el *SIW* de la Farmacia del hospital). El *RA* utiliza el conocimiento que tiene almacenado en su *Base de Conocimiento* sobre los *Sistemas de Información* para contactar aquel que podría responder las consultas. Este caso es una especialización del *primero*.

Tercero, la consulta ha sido dividida en varias subconsultas en la etapa de análisis. El *RA* analiza qué agentes podrían responder cada una de las subconsultas y con ellos compone la red de vecinos. El proceso aplicado para seleccionar las fuentes de información (*ISAgents*) para cada consulta es el mismo definido en el primer caso. Finalmente, la red de vecinos se compone de la unión de las diferentes subredes generadas para cada subconsulta.

Para la consulta del *Doctor Smith*, el *RA* selecciona como fuentes de información el *SIW* de la *Farmacia*, el del *Laboratorio Clínico* y el de los *Nutricionistas* del hospital. El *RA* también genera las tres consultas que redirigirá a los *ISAgents* de dichos *SIW*. Vale la pena mencionar que dichas consultas también contienen las preferencias del usuario como se muestra a continuación:

(*deffacts Consulta1* (*IDUsuario "Doctor John Smith"*)
(*ISAgent "LabClínicoISA") (info_requerida "análisis sanguíneos"*)
(*acción mostrar*) (*atributo (nombre formato_gráfico*) (*lista "JPEG"*))
(*problema* (*nombre FormatoNoSoportadoporDM*) (*tipo incompatibilidad*) (*causas SoloSoportaTexto*))
(*atributo* (*nombre formato_texto*) (*lista "XML" "txt"*)))

(*deffacts Consulta2* (*IDUsuario "Doctor John Smith"*)
(*ISAgent "NutricionistasISA"*) (*info_requerida "dieta"*)
(*acción mostrar*) (*atributo* (*nombre formato_gráfico*) (*lista "JPEG"*))
(*problema* (*nombre FormatoNoSoportadoporDM*) (*tipo incompatibilidad*) (*causas SoloSoportaTexto*))
(*atributo* (*nombre formato_texto*) (*lista "XML" "txt"*)))

(*deffacts Consulta3* (*IDUsuario "Doctor John Smith"*)
(*ISAgent "FarmaciaISA"*) (*info_requerida "medicamentos prescritos"*)
(*acción mostrar*)(*atributo* (*nombre formato_gráfico*) (*lista "JPEG"*))
(*problema* (*nombre FormatoNoSoportadoporDM*) (*tipo incompatibilidad*) (*causas SoloSoportaTexto*))
(*atributo* (*nombre formato_texto*) (*lista "XML" "txt"*)))

El *RA* debe también analizar el nivel de confianza asociado a estos vecinos (por ejemplo, el *SIW* más cercano). Si es la primera vez que el *RA* ejecuta esta consulta o que trabaja con estos *ISAgents*, el *RA* les envía la consulta a través de un mensaje transmitido en *broadcast*.

7.3 Redirección de la consulta

Una vez el *RA* ha identificado las fuentes de información potenciales (vecinos), redirecciona la consulta mediante el envío de un mensaje a sus vecinos (que incluye la consulta). El *RA* puede usar un mensaje orientado (para receptores específicos) o un mensaje *broadcast* a todos los vecinos (por ejemplo, esperando el primero en responder, u obteniendo todas las respuestas y luego analizando cuáles de ellas son más confiables). Si el *RA* contempla un esquema de confianza para los agentes que componen la red de vecinos, este agente podría enviar el mensaje secuencialmente, comenzando por el vecino más confiable. Si este responde, el proceso se termina. De lo contrario, el *RA* debe continuar el envío de mensajes hasta que haya contactado al agente menos confiable (acorde con las ideas de Agostini *et al.* [1]).

Si el *RA* sabe que el *vecino1* puede responder la *consulta1*, que el *vecino2* la *consulta2* y así sucesivamente, envía un mensaje orientado a cada uno de los vecinos (basado en el hecho (1), ver sección 5.1). Por ejemplo, para la consulta del *Doctor Smith*, el *RA* envía la *Consulta1* al *"LabClínicoISA",* la *Consulta2* al *"NutricionistasISA"* y la *Consulta3* al *"FarmaciaISA"*.

Finalmente, el *RA* debe recopilar las respuestas obtenidas de los diferentes agentes y seleccionar las más relevantes de acuerdo a los criterios establecidos (como se explicó en la sección 7.2).

8 Impacto de los cambios de localización

En esta sección, mostramos cómo la composición de la *red de vecinos* (actividad 2 del *Query Routing*, ver sección 7.2) podría ser altamente dinámica ya que las consultas son de alguna manera dependientes de la localización. Por está razón, se debe tener en cuenta que la localización de los vecinos o la de la fuente de la consulta puede cambiar (por ejemplo, cuando la consulta debe ser respondida por agentes que se encuentren a una distancia no superior a una distancia dada del emisor de la consulta y estos pueden cambiar de localización). Para propósitos de enrutamiento, podemos considerar tres casos de conformación de la red de vecinos: *Primero*, tener en cuenta sólo la localización del emisor o fuente de la consulta (los receptores son fijos y se debe considerar sólo los cambios ocurridos en la localización de la fuente de la consulta). *Segundo*, tener en cuenta sólo la localización de los receptores de la consulta (el emisor o fuente de la consulta es fijo y sólo se consideran los cambios en la localización de los receptores de la consulta). *Tercero*, tener en cuenta tanto los cambios de localización del emisor como de los receptores de la consulta.

Para conocer la localización del agente emisor de la consulta, el *RA* puede consultar al *Coordinator Agent* (del *SMA* de *Comunicación*) que conoce todos los agentes y sus servicios ofrecidos. Con el fin de conocer la localización de los agentes receptores de la consulta, el *RA* puede consultar los *ISAgents.*

Para las consultas basadas sólo en la localización del emisor (primer caso), es también necesario conocer tanto su orientación como su velocidad de desplazamiento. El *Proxy Agent* (del *SMA* de *Comunicación*) solicita las características de la localización al *Connection Controller Agent* (del *SMA* de *Conexión*). Este último agente pregunta la localización del emisor al *Mobile Device Agent* (agente que se ejecuta en el *DM* del usuario) que a su turno verifica los cambios de localización del emisor de la consulta. El *Connection Controller Agent* le envía un mensaje al *Mobile Device Agent* en el que le pregunta si los resultados que van a ser desplegados son aún válidos considerando su nueva localización. Si es así, la red de vecinos no cambia. De otra manera, el *RA* debe conformar otra red de vecinos basada en la nueva localización del emisor de la consulta.

En el segundo caso, (consultas basadas en la localización de los receptores de la consulta), la red de vecinos puede solamente organizarse justo después que el *RA* ha consultado a los *ISAgents* acerca de la localización. En este caso, es necesario tomar en cuenta la dirección y velocidad de desplazamiento de los agentes receptores.

En el último caso, (consultas basadas en los cambios de localización tanto para el emisor como el receptor de la consulta), debemos considerar las soluciones dadas para los dos primeros casos.

9 Implementación de PUMAS

Nuestro trabajo apunta a la implementación y prueba de cada uno de los *SMA* de *PUMAS.* Para esto, hemos escogido *JADE-LEAP* (*http://jade.tilab.com/*), una plataforma acorde con los estándares *FIPA*. Hemos probado los ejemplos que vienen con esta plataforma en dos tipos diferentes de *DM* de nuestro equipo de investigación (*Pocket PCs* con *Windows CE*, que utiliza *Crème* – kVM, acorde con la implementación *Personal Java* - y dos *PDA* con *PalmOS* utilizando una implementación *MIDP* 1.0). De dichas pruebas, hemos encontrado algunas correspondencias entre nuestros agentes y aquellos que ofrece *JADE-LEAP*: Podemos asociar su *AMS* (*Agent Management System*) con nuestro *Coordinator Agent* (*CA*), el *DF* (*Directory Facilitator*) con nuestros *MDProfile Agent*, *Receptor/Provider Agent* y *Router Agent.* Así mismo, el *ACC* (*Agent Communication Channel*) con nuestro *Connection Controller Agent* (*CCA*). Para la representación de conocimiento, *JADE-LEAP* usa *JESS* y un conjunto de clases (*JADE Content Package*) que permiten la definición de ontologías. Es por esta razón que la sección 5 de este artículo está consagrada a la definición y representación en *JESS* del conocimiento de los agentes de *PUMAS*. *JADE-LEAP* también permite la definición de ontologías propias, razón por la cual queremos

integrar *OWL* para llevar a cabo esta actividad. En nuestras pruebas de los ejemplos de *JADE-LEAP*, encontramos algunos problemas como la inconsistencia entre los agentes activos listados en la plataforma y, los agentes conectados de acuerdo a la aplicación (algunos agentes desconectados aparecían como conectados en la plataforma). De allí surgió la necesidad de crear el *Connection Controller Agent* (que controla todos los agentes conectados) y el *Coordinator Agent* (que mantiene un registro de los agentes conectados al sistema y que puede jugar el rol del *Connection Controller Agent* si éste falla). Los ejemplos fueron reformados y dichas inconsistencias se resolvieron. Estas reformas son la base para la construcción del *SMA* de *Conexión*. Finalmente, utilizamos la forma de ejecución de *JADE-LEAP* conocida como *"Split Execution"* donde una parte de la plataforma se ejecuta de manera centralizada (lo que llamamos la plataforma central de *PUMAS*) y la otra en los *DM* (en nuestro caso, los *Mobile Device Agents*). Esta manera de ejecución simula un sistema *P2P Híbrido* con las siguientes ventajas: conocimiento de los agentes activos, sus servicios y capacidades; menos tráfico en la red (son locales los mensajes de los agentes de la plataforma central de *PUMAS*). Finalmente se mantiene un mecanismo de registro de los agentes (suscripción) y de autenticación que permite a un agente reconocer los otros agentes con los que puede interactuar y a los que puede solicitar servicios.

10 Conclusiones y trabajo futuro

En este artículo, hemos descrito el conocimiento manejado e intercambiado por los agentes de los *SMA* de *Información* y de *Adaptación* de *PUMAS* con el fin de llevar a cabo la adaptación de la información. *PUMAS* es un *framework* que recupera a partir de diferentes *SIW*, información adaptada tanto al perfil de usuario (que contiene sus necesidades, características, preferencias, historia en el sistema, localización actual, etc.) como a las capacidades técnicas del *DM* que dicho usuario utiliza para acceder los *SIW*. También hemos descrito las estrategias seguidas por el *Router Agent* con el fin de llevar a cabo el proceso de *enrutamiento de consultas* (*Query Routing*). En *PUMAS*, dicho proceso se compone de tres actividades: el análisis de la consulta, la selección de fuentes de información y la redirección de la consulta. Presentamos cada una de estas actividades y las ilustramos a través de un *SIW* de un hospital. Mostramos de igual manera el impacto que tienen los cambios de localización (del emisor y/o receptor de las consultas) en el proceso de *Query Routing*, particularmente en la composición de la red de vecinos, principal tarea de la actividad de selección de las fuentes de información. Finalmente, mostramos las decisiones tomadas para la implementación de *PUMAS*.

Nuestro trabajo futuro se orienta a la definición de una extensión de un *ACL* (*Agent Communication Language*) que considere las características nómadas del usuario (es decir, introducir primitivas tales como *query-where*, *query-when* con el fin de dirigir consultas sólo a usuarios cercanos o, durante un determinado periodo de tiempo). También va encaminado hacia la elección de los mecanismos y algoritmos para cada una de las actividades del proceso de *Query Routing*. Nuestro objetivo final es completar la implementación de *PUMAS* con sus correspondientes pruebas.

Agradecimientos. La autora *Angela Carrillo Ramos* agradece a *Fernando De la Rosa* y *Alexandra Méndez Lindo* por los comentarios efectuados a este trabajo.

Referencias

[1] A. Agostini y G. Moro. Identification of Communities of Peers by Trust and Reputation. Proc. of the 11th International Conference in Artificial Intelligence: Methodology, Systems, and Applicacións (AIMSA 2004) (Varna, Bulgaria, September 2-4, 2004). LNCS, Vol. 3192. Springer-Verlag, Berlin Heidelberg, (2004), pp. 85-95.

[2] S. Albayrak, S. Wollny, N. Varone, A. Lommatzsch y D. Milosevic. Agent Technology for Personalized Information Filtering: The PIA-System. Proc. of 20th Annual ACM Symposium on Applied Computing (SAC 2005) (Santafe, New Mexico, USA, March 13-17, 2005). ACM Press, New York, NY (2005), pp. 54-59.

[3] A. Carrillo Ramos, J. Gensel, M. Villanova-Oliver y H. Martin. Modelling with Ubiquitous Agents a Web-based Information System Accessed through Mobile Devices. Proc. of the Cooperative Inf. Systems (CoopIS'04). (Larnaca, Cyprus, October 25-29, 2004). LNCS, Vol. 3290, Springer-Verlag, Berlin Heidelberg (2004), pp. 264-282.

[4] F. Gandon y N. Sadeh. Semantic Web Technologies to Reconcile Privacy and Context Awareness. Journal of Web Semantics. Vol. 1, Issue 3. (October 31, 2004). *http://www.websemanticsjournal.org/ps/pub/2004-17*. (Ultima revisión: Marzo 2005)

[5] J. Indulska, R. Robinson, A. Rakotonirainy, y K. Henricksen. Experiences in Using CC/PP in Context-Aware Systems. Proc. of Mobile Data Management: 4th International Conference (MDM 2003) (Melbourne, Australia, January 21-24, 2003), LNCS, Vol. 2574. Springer-Verlag, Berlin Heidelberg (2003), pp. 247-261.

[6] F. Koch e I. Rahwan. Classification of Agent-based Mobile Assistant. Proc. of Workshop on Agents for Ubiquitous Computing (UbiAgents04) (NY City, USA July 20, 2004). *http://www.ift.ulaval.ca/~mellouli/ubiagents04/* (Ultima revisión: Marzo 2005)

[7] K. Kurumatani. Mass User Support by Social Coordination among Citizen in a Real Environment. Proc. of Multi-Agent for Mass User Support. International Workshop (MAMUS 2003). (Acapulco, Mexico, August 10, 2003). LNAI, Vol. 3012, Springer-Verlag, Berlin Heidelberg (2003), pp. 1–16.

[8] J. Odell, H. Van Dyke Parunak y B. Bauer. Representing Agent Interaction Protocols in UML. Proc of Agent Oriented Sw Engineering (AOSE 2000) (Limerick, Ireland, June 10, 2000), LNCS, Vol. 1957. Springer-Verlag Berlin Heidelberg, (2000), pp.121-140.

[9] J. Park y S. Barber. Finding Information Sources by Model Sharing in Open Multi-Agent System. Proc. of Workshop on Agents for Ubiquitous Computing (UbiAgents04) (NY, USA, July 20, 2004) *http://www.ift.ulaval.ca/~mellouli/ubiagents04/* (Ultima revisión: Marzo 2005)

[10] M. Pirker, M. Berger y M. Watzke. An approach for FIPA Agent Service Discovery in Mobile Ad Hoc Environments. Proc. of Workshop on Agents for Ubiquitous Computing (UbiAgents04) (NY, USA July 20, 2004). *http://www.ift.ulaval.ca/~mellouli/ubiagents04/* (Ultima revisión: Marzo 2005)

[11] J. Xu, E. Lim y W.K. Ng. Cluster-Based Database Selection Techniques for Routing Bibliographic Queries. Proc. of the 10th Int. Conference and Workshop on Database and Expert Systems Applications (DEXA 99) (Florence, Italy, August 30-September 3, 1999), LNCS, Vol. 1677. Springer-Verlag, Berlin Heidelberg, pp. 100-109.

[12] D. Yang, L. Xu, W. Cai, S. Zhou y A. Zhou. Efficient Query Routing for XML Documents Retrieval in Unstructured Peer to Peer Networks. Proc. of the 6th Asia Pacific Web Conference (APWeb 2004) (Hangzhou, China, April 14-17, 2004). LNCS, Vol. 3007, Springer-Verlag, Berlin Heidelberg, (2004), pp.217-223.

[13] *http://www.w3.org/TR/webont-req/* (Ultima revisión: Marzo 2005)