Plataforma de descubrimiento de servicios para ambientes de computación ubicua basada en preferencias de usuario, especificaciones de dispositivos y contexto de entrega[1]

Service Discovery Platform for Ubiquitous Computing Environments based on User Preferences, Device Specifications and Delivery Context[2]

Plataforma de descobrimento de serviços para ambientes de computação ubíqua baseada em preferências de usuário, especificações de dispositivos e contexto de entrega[3]

Juan Carlos Corrales-Muñoz[4]
Luis Javier Suárez-Meza[5]
Luis Antonio Rojas-Potosí[6]

[1] Fecha de recepción: 5 de julio de 2010. Fecha de aceptación: 10 de marzo de 2011. Este artículo se deriva de un proyecto de investigación denominado *Desarrollo del producto iCom Centrex IP sobre una arquitectura de servicios convergentes soportada en tecnologías abiertas,* desarrollado por el grupo de Ingeniería Telemática (GIT) y financiado por Colciencias, Avatar Ltda. y la Universidad del Cauca, Popayán, Colombia.

[2] Submitted on: July 5, 2010. Accepted on: March 10, 2011. This article results from the research project *Developing the Product iCom Centrex IP on a Convergent Service Architecture Based on Open Technologies,* developed by the research group GIT and financed by Colciencias, Avatar Ltd and the Universidad del Cauca, Popayán, Colombia.

[3] Data de recepção: 5 de julho de 2010. Data de aceitação: 10 de março de 2011. Este artigo se deriva de um projeto de pesquisa denominado *Desenvolvimento do produto iCom Centrex IP sobre una arquitetura de serviços convergentes suportada em tecnologias abertas,* desenvolvido pelo grupo de Engenharia Telemática (GIT) e financiado por Colciencias, Avatar Ltda. e pela Universidad do Cauca, Popayán, Colômbia.

[4] Ingeniero en electrónica y telecomunicaciones, Universidad del Cauca, Popayán, Colombia. Magíster en Ingeniería Telemática, Universidad del Cauca. PhD in Computer Science, University of Versailles Saint-Quentin-en-Yvelines, Francia. Docente y coordinador del Grupo de Ingeniería Telemática, Cauca, Colombia. Correo electrónico: jcorral@unicauca.edu.co.

[5] Ingeniero en electrónica y telecomunicaciones, Universidad del Cauca, Popayán, Colombia. Miembro del Grupo de Ingeniería Telemática, Universidad del Cauca. Correo electrónico: ljsuarez@unicauca.edu.co.

[6] Ingeniero en electrónica y telecomunicaciones, Universidad del Cauca, Popayán, Colombia. Miembro del Grupo de Ingeniería Telemática, Universidad del Cauca. Correo electrónico: luisrojas@unicauca.edu.co.

Resumen

El buscar mejoras en la capacidad de entregar servicios que cumplan con las solicitudes del usuario ha llevado a desarrollar diversos proyectos orientados al descubrimiento de servicios y a entenderlo como una fase importante dentro del proceso de composición. Este descubrimiento ha tenido que adaptarse para generar una búsqueda que satisfaga la actual diversidad de dispositivos utilizados para acceder a los servicios y el aumento de sus capacidades, mostrando un avance hacia el concepto de computación ubicua. El presente artículo propone un mecanismo que, utilizando los conceptos de preferencias de usuario, especificaciones de dispositivos y contexto de entrega, oriente el descubrimiento de servicios a entornos ubicuos. Así, el mecanismo de descubrimiento propuesto trabaja sobre procesos BPEL, que representan los servicios disponibles en la red ubicua, abstraídos a una representación formal de grafos, a fin de trasladar el problema del emparejamiento de archivos BPEL a un emparejamiento de grafos. De forma que es posible obtener un emparejamiento aproximado si no existe un servicio que corresponda exactamente con los requisitos del usuario.

Palabras clave

Computación ubicua, servicios a usuarios de información, usuarios de información.

Abstract

The search for improvements in the ability to deliver services that meet user requests has led to the development of numerous projects aimed at service discovery, which is an important stage in the composition process. This discovery has required adaptations aimed at finding suitable devices for accessing services and increasing its capacity. This has led to advances toward the concept of ubiquitous computing. This paper presents a mechanism which uses the concepts of user preferences, device specifications, and delivery context, in order to assist the discovery of services in ubiquitous environments. The proposed discovery mechanism works based on BPEL processes, which represents the services available in the ubiquitous network. They are abstracted to a formal representation of Graphs in order to move the BPEL files matching problem towards Graphs matching. This makes it possible to obtain an approximate matching in the absence of a service that totally matches user requirements.

Key words

Ubiquitous computing, services for the users of information, information users.

Resumo

Procurar melhorias na capacidade de entregar serviços que cumpram com as solicitações do usuário tem levado ao desenvolvimento de diversos projetos orientados ao descobrimento de serviços e a entendê-lo como uma fase importante dentro do processo de composição. Este descobrimento vem sendo adaptado para gerar uma busca que satisfaça a atual diversidade de dispositivos utilizados para acessar os serviços e o aumento de suas capacidades, mostrando um progresso na direção do conceito de computação ubíqua. Este artigo propõe um mecanismo que, utilizando os conceitos de preferências de usuário, especificações de dispositivos e contexto de entrega oriente o descobrimento de serviços a ambientes ubíquos. Dessa forma, mecanismo de descobrimento proposto trabalha sobre processos BPEL, que representam os serviços disponíveis na rede ubíqua, abstraídos a uma representação formal de grafos, com o objetivo de trasladar o problema do emparelhamento de arquivos BPEL a um emparelhamento de grafos. De forma que é possível obter um emparelhamento aproximado se não existe um serviço que corresponda exatamente com os requisitos do usuário.

Palavras chave

Computação ubíqua, serviço a usuários de informação, usuários de informação.

Introducción

Debido a la evolución de las tecnologías de comunicaciones y a la gran diversidad de dispositivos móviles que han emergido en los últimos años, que respaldan cada vez más mejores servicios, en la actualidad se evidencia un uso masivo de terminales distribuidos, configurables dinámicamente y en proximidades a los usuarios que permiten un acceso permanente a la información. Ello abre paso a la visión introducida por Weiser (1991) con el nombre de *computación ubicua*. Según este autor, la computación ubicua se describe como la existencia de pequeños computadores, con capacidades de comunicación y computación, embebidos de forma casi invisible en cualquier tipo de dispositivo cotidiano, que se integran amigablemente con los humanos. Es decir, las personas interactúan con ellos de forma inconsciente (Almenárez, 2005; Weiser, 1991).

Dado que en estos ambientes las personas están centradas en las tareas que deben cumplir, más que en el dispositivo que deben utilizar para ejecutarlas, ya que los equipos pasan inadvertidos, uno de los objetivos de la ubicuidad se enfoca en ayudar a los usuarios a identificar, en cualquier momento y en cualquier lugar, las tareas que se van a realizar, por medio del descubrimiento automático de los servicios ofrecidos en la red ubicua (Ben Mokhtar et ál., 2006).

Sin embargo, encontrar un servicio que cumpla exactamente con estas peticiones se convierte en un caso excepcional, ya que la mayoría requieren una adaptación de los servicios disponibles en la red o un proceso de composición de diversas capacidades para ejecutar tareas complejas. De este modo, en contextos tan dinámicos y heterogéneos como los ambientes ubicuos, es imprescindible contar con mecanismos de descubrimiento de servicios que consideren las preferencias del usuario, especificaciones de dispositivos y contexto de entrega, que ofrezcan la suficiente flexibilidad para reconfigurar los servicios de acuerdo con las variaciones del ambiente.

El descubrimiento de servicios deberá incorporar las características necesarias para ser tan dinámico y autónomo como las condiciones de la red ubicua lo

requieran. Por ello la personalización es una de característica muy importante dentro de este proceso.

El paradigma de la personalización se enfoca en adaptar aplicaciones tanto como sea posible a las preferencias y al contexto del usuario. Hace referencia al conjunto de técnicas que permiten proveer al usuario información relevante (Abbar et ál., 2008). La necesidad de tener en cuenta la movilidad y la omnipresencia (Belotti et ál., 2005) ha impuesto nuevas consideraciones, como la localización de usuario, el medio utilizado para la interacción y muchas otras características agrupadas en el concepto de *contexto de usuario*.

En este artículo se argumenta que la personalización debe ser considerada dentro del proceso de descubrimiento de servicios y, además, dicho proceso, requiere una fase de emparejamiento atómico que opere sobre modelos de comportamiento, procesos tipo *Business Process Execution Language* (BPEL) (Andrews, 2010). Típicamente, BPEL se usa como un lenguaje de orquestación de servicios, con el objetivo de formar procesos de negocio ejecutables. En la actualidad, el número de procesos de negocio descritos en BPEL sobre la web y en el ámbito empresarial se ha incrementado considerablemente. Así mismo, BPEL es útil para formar una composición de múltiples servicios que cumpla con los requerimientos del usuario cuando un único servicio no puede realizar la tarea requerida (Corrales, 2008).

Así, el presente artículo expone un mecanismo de recuperación de servicios basado en modelos de comportamiento, que considera tanto las preferencias como el contexto del usuario, y cuenta con una etapa de emparejamiento que permite evaluar la distancia semántica entre los servicios presentes en la red ubicua y los requeridos por el usuario. De esta forma, si no existe un servicio que se ajuste exactamente a las peticiones del cliente, se recuperan y postulan los servicios semánticamente más similares. Para hacer esto, se ha utilizado una representación formal de grafos, con el objetivo de transformar el problema del emparejamiento de procesos descritos en BPEL a un problema matemático de emparejamiento de grafos, adaptando algoritmos existentes para este propósito.

1. Trabajos relacionados

El *descubrimiento de servicios* se define como la capacidad de encontrar y utilizar posteriormente un servicio basado en alguna descripción publicada de su funcionalidad y parámetros operacionales (Bandara et ál., 2007). El descubrimiento de servicios puede abordarse bajo dos enfoques principalmente: descubrimiento sintáctico y semántico.

El descubrimiento sintáctico se basa en técnicas de comparación de interfaces (por ejemplo, UDDI, WSDL, IDL, interfaces RMI, etc.) o palabras clave para buscar servicios. Ello requiere coincidencias exactas sintácticas entre las descripciones de los servicios y los parámetros empleados (Ben Mokhtar et ál., 2007; Corrales, 2008).

La descripción semántica de servicios busca la posibilidad de que las máquinas realicen un mayor procesamiento y razonamiento sobre los servicios. La principal diferencia con las descripciones sintácticas es la forma como se representa la información. Mientras la sintáctica define los servicios a partir de los mensajes de entrada y salida, tipos y partes de los mensajes, la semántica ofrece información acerca de la funcionalidad del servicio (Ben Mokhtar et ál., 2007; Corrales et ál., 2008). La representación semántica del contenido de las descripciones de un servicio permiten a las máquinas entender y procesar su contenido, a la vez que respalda el descubrimiento e integración dinámica de los servicios (Ben Mokhtar et ál., 2007).

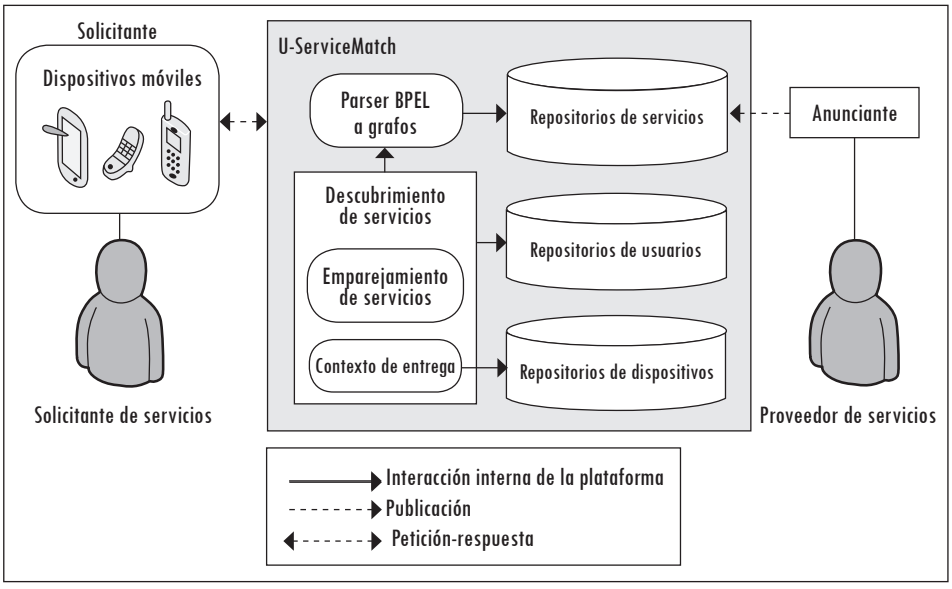
Así mismo, el descubrimiento de servicios se clasifica en *centralizado vs. descentralizado* y *sintáctico vs. semántico*. Actualmente, se observa que los enfoques descentralizados y semánticos son los más adecuados para entornos web y su extensión a ambientes ubicuos (Sellami et ál., 2009; Bandara et ál., 2007); pero la aplicación de razonadores semánticos en ambientes ubicuos aumenta la cantidad de procesamiento requerido para una respuesta, que incremente considerablemente el tiempo de ejecución. Así, este es uno de los factores críticos en este tipo de entornos (Steller y Krishnaswamy, 2009).

De lo anterior, en el presente artículo se propone un enfoque de descubrimiento de servicios para entornos ubicuos basado en un emparejamiento semántico, sin usar razonadores, que proporciona un *ranking* de servicios que cumple total o parcialmente con la solicitud del usuario. Además, dentro del proceso de descubrimiento se consideran las preferencias de usuario, las especificaciones/capacidades de los dispositivos de acceso y su contexto de entrega, con el objetivo de proporcionar flexibilidad para configurar los servicios de acuerdo con los cambios del entorno.

2. Plataforma de descubrimiento de servicios

En esta sección describimos la arquitectura de la plataforma de descubrimiento de servicios en ambientes de computación ubicua (Figura 1). El objetivo de esta plataforma es proveer un mecanismo de recuperación de servicios en ambientes de computación ubicua, que tenga en cuenta tanto las preferencias del usuario como las especificaciones de dispositivos y el contexto de entrega.

Figura 1. Arquitectura de U-ServiceMatch



Fuente: presentación propia de los autores.

El usuario interactúa con la plataforma por medio de una aplicación que respalda el envío de las peticiones expresadas como modelos de comportamiento BPEL. La plataforma recibe las peticiones y las transforma a una representación formal de grafos para ejecutar la recuperación de servicios. Esta es mejorada capturando el contexto de entrega del usuario. Igualmente, el perfil del usuario es recuperado y comparado con otros perfiles para disminuir el tiempo de respuesta del sistema. Finalmente, la plataforma retorna los servicios más similares a la consulta realizada por el usuario.

Por otro lado, los proveedores publican sus servicios como procesos BPEL, los cuales son almacenados en el repositorio de servicios y puestos a disposición para ser consumidos. A continuación se describen brevemente cada uno de los módulos que componen la plataforma:

- *Solicitante*: corresponde a los clientes móviles que interactúan con la plataforma por medio su navegador web (*browser*), con el fin de consultar, invocar y consumir los servicios de la red ubicua.
- *Anunciante:* este módulo ofrece las capacidades necesarias para que los proveedores publiquen sus servicios en el repositorio de la plataforma. Las descripciones de servicios contienen atributos funcionales, como entradas, salidas,

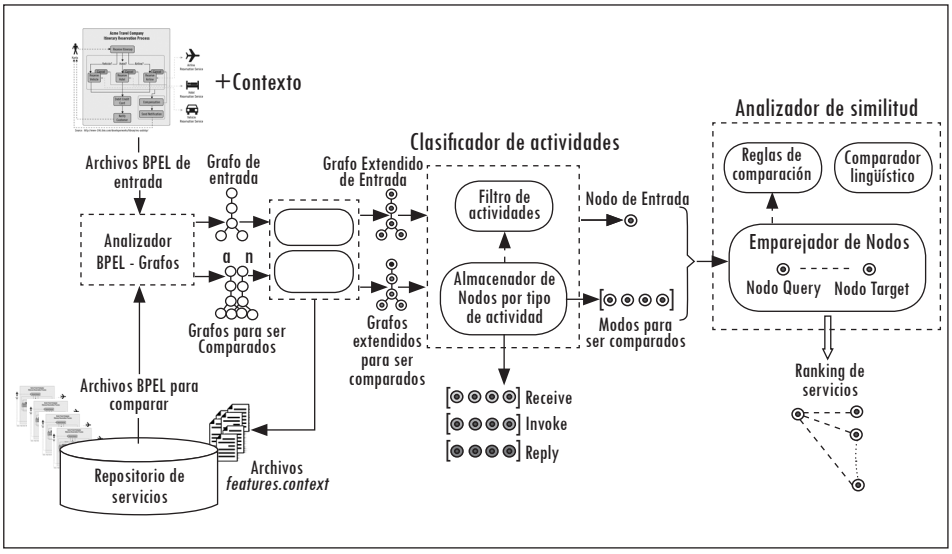
precondiciones, restricciones, etc. Este módulo es implementado mediante una aplicación web.

- *Parser BPEL a grafos:* este módulo permite transformar las descripciones de comportamiento BPEL en su equivalente en grafos. El algoritmo emplea un proceso de transformación recursivo para cada tipo de actividad estructurada tomando una aproximación de arriba-abajo (*top-down*). Las actividades básicas BPEL (*invoke*, *receive* y *reply*) son transformadas en nodos y las secuencias son obtenidas conectando los nodos requeridos por medio de aristas. Las actividades estructuradas son representadas por medio de operadores lógicos XOR y AND (Corrales, 2008).
- *Módulo de descubrimiento de servicios:* basado en un algoritmo de emparejamiento de actividades básicas BPEL (módulo de *emparejamiento de servicios*) y considerando tanto las preferencias del usuario (repositorio de usuarios) como las capacidades de los terminales móviles (*repositorio de dispositivos*) y las restricciones del contexto del solicitante (módulo de *contexto de entrega*), este módulo recupera los servicios más similares del *repositorio de servicios*, como respuesta a una consulta definida por el anunciante. En las siguientes secciones presentamos con detalle las funcionalidades de cada uno de los módulos que lo componen.

2.1 Emparejamiento de servicios

En esta sección mostramos cómo el emparejamiento atómico de procesos BPEL es reducido a un emparejamiento de grafos, con el fin de obtener los servicios más pertinentes para el usuario. También, se presenta el mecanismo de consulta que permite obtener la información del contexto para que el servicio pueda ser consumido (Figura 2).

Figura 2. Emparejamiento de actividades básicas BPEL



Fuente: presentación propia de los autores.

Los módulos *Adicionar Contexto* y el *Motor de Consulta* implementan el mecanismo de consulta del metadato *features.context*, que contiene los requerimientos de contexto del servicio, los cuales son descritos a continuación:

- *Motor de consulta*: es un motor que obtiene las URI de los archivos BPEL almacenados en el repositorio de procesos de negocio y la información contenida dentro del archivo *features.context*.
- *Adicionar contexto*: este módulo aprovecha la flexibilidad que la representación formal basada en grafos ofrece. Una vez el *motor de consulta* obtiene la información de contexto del servicio del archivo *features.context*, asociado al documento BPEL, este adiciona el atributo *AccessType* al grafo, haciendo que los nodos que lo componen posean un parámetro adicional (*ActivityType, Operation, PortType, PartnerLink* y *AccessType*). La utilidad de este nuevo atributo se evidencia después del proceso de emparejamiento, ya que permite filtrar los servicios que no pueden ser consumidos por el dispositivo móvil.

Antes de ejecutar el algoritmo que empareja las actividades básicas de los grafos (G_Q y G_T), es necesaria una etapa previa, encargada de clasificar, organizar y filtrar los nodos que los componen, de acuerdo con su tipo de actividad (*módulo clasificador de actividades*). De esta forma, sólo los nodos que pertenecen al mismo tipo de actividad en G_Q (grafo de entrada) y G_T (grafo que va a ser comparado),

respectivamente, son comparados, es decir: el conjunto de actividades $Invoke_{syn}$, $Invoke_{asyn}$, *Receive* y *Reply*.

Una vez obtenidos y organizados los nodos, tanto de G_Q como de G_T, el módulo *Analizador de Similitud* permite compararlos considerando las *reglas de comparación*, función definida mediante el Algoritmo 1 y el *Comparador Lingüístico*, definido por la función *LinguisticSimilarity* (LS). El *Emparejador de Nodos* toma como entradas dos nodos, y calcula la distancia semántica entre ellos. Como se mencionó, cada nodo posee cinco atributos: *ActivityType* (AT), *Operation* (Op), *PortType* (PT), *PartnerLink* (PL) y *AccessType* (AccT), de los cuales tres son considerados en el proceso de emparejamiento (Op, PT y PL), ya que el parámetro AT permite identificar que los nodos que se van a comparar sean del mismo tipo y el AccT permite filtrar los servicios que no pueden ser consumidos por el dispositivo de acceso.

La función *BasicActivityMatch*¸ presentada en el Algoritmo 1, inicia dando prioridad a la comparación de Op. Si las dos operaciones son similares (*SimOperation*$>$ 0), se continua con cálculo de la similitud de los demás parámetros (PT y PL) para estimar la distancia entre las dos actividades, *DistanceNode*. Los pesos *Wop*, *Wpt* y *Wpl* indican la contribución de la similitud de los atributos Op, PT y PL a la similitud de las actividades ($0 \leq Wop \leq 1$, $0 \leq Wpt \leq 1$ y $0 \leq Wpl \leq 1$). En el cálculo de la similitud de los atributos se emplea la función *LS*, que calcula la similitud lingüística entre dos etiquetas basándose en sus nombres. Para obtener esta medida se utilizan los algoritmos *NGram*, *CheckSynonym* y *CheckAbbreviation* (Corrales et ál., 2010).

Algoritmo 1. Emparejamiento de actividades básicas (*BasicActivityMatch*)

INPUTS: ($Node_i$, $Node_j$) $Node_i$: Struct (Op_i, PT_i, PL_i, AT_i), $Node_j$: Struct (Op_j, PT_j, PL_j, AT_j)
OUTPUT: DistanceNode
BEGIN
Calculate Operation Similarity $SimOperation = LS(Op_i, Op_j)$
if SimOperation = 0 (different Operations) **then**
return DistanceNode = 1
else
Calculate PortType Similarity $SimPortType = LS(PT_i, PT_j)$
Calculate PartnerLink Similarity $SimPartnerLink = LS(PL_i, PL_j)$
CalculateDistanceNode

$$DistanceNode = 1 - \frac{w_{op}*SimOperation + w_{pt}*SimPortType + w_{pl}*SimPartnerLink}{w_{op} + w_{pt} + w_{pl}}$$

1. **end if**
2. **return** DistanceNode
3. **END**

El algoritmo *NGram* estima la similitud de acuerdo con el número común de *qgramas* entre las etiquetas (Angell et ál., 2002). El algoritmo *CheckSynonym* utiliza el diccionario lingüístico WordNet (Miller, 1995) para identificar sinónimos, mientras el algoritmo *CheckAbbreviation* usa un diccionario de abreviaciones adecuado al dominio de aplicación. Si todos los algoritmos entregan un valor de uno, existe una coincidencia exacta entre las etiquetas; si entregan un valor de cero, no hay similitud entre las palabras. Si los valores entregados por *Ngram* y *CheckAbbreviation* son iguales a cero y el valor de *CheckSynonym* está entre cero y uno, el valor total de la similitud es igual a *CheckSynonym*. Finalmente, si los tres algoritmos arrojan un valor entre cero y uno, la similitud lingüística es el promedio de los tres. La función *LS* se describe a continuación.

$$\mathrm{LS} = \begin{cases} 1 \text{ si } (\mathrm{m1} = 1 \vee \mathrm{m2} = 1 \vee \mathrm{m3} = 1) \\ \mathrm{m2} \text{ si } (0 < \mathrm{m2} < 1 \wedge \mathrm{m1} = \mathrm{m3} = 0) \\ 0 \text{ si } (\mathrm{m1} = \mathrm{m2} = \mathrm{m3} = 0) \\ \dfrac{\mathrm{m1} + \mathrm{m2} + \mathrm{m3}}{3} \text{ si } (\mathrm{m1},\mathrm{m2},\mathrm{m2} \in (0,1)) \end{cases}$$

Donde: $m1 = \mathrm{Sim}(NGram), m2 = \mathrm{Sim}(CheckSynonym), m3 = \mathrm{Sim}(CheckAbbreviation)$.

2.2 Contexto de entrega

Para la gestión del contexto hemos definido dos parámetros: las restricciones del contexto del solicitante y los requerimientos del contexto del servicio. Dichas variables se sustentan en el *Repositorios de Dispositivos* y el *Repositorio de Servicios*, respectivamente. A continuación se describe la función *CheckDeliveryContext*, la cual permite obtener durante el proceso de descubrimiento, el contexto tanto del usuario como del proveedor de servicios (Algoritmo 2).

Algoritmo 2. Función verificar contexto (*CheckDeliveryContext*)

```
INPUTS: (listRankedServices)
OUTPUT: rankedFilteredServices
BEGIN
Device Profile deviceProfile = lookupDeviceProfile(deviceURL)
for each node in listRankedServices do
EmfContextfeatures.context = lookupFeatures.context(node.URI)
for each networkaccess in features.context do
if deviceProfile contains networkaccess (network access supported) then
Add node to rankedFilteredServices
break for
end if
end for
end for
return rankedFilteredServices
END
```

La función *CheckDeliveryContext* toma el *ranking* de los servicios obtenidos en la fase de emparejamiento de actividades básicas y verifica los requerimientos de contexto de los servicios recuperados del *Repositorio de Servicios*. Las restricciones de contexto del usuario son determinadas a través del *Repositorio de Dispositivos* mediante las cabeceras HTTP, UAProf y WURFL. Así, las características de los dispositivos son relacionadas con las restricciones del proveedor de servicios. En primer lugar, el perfil del dispositivo es obtenido desde el *Repositorio de Dispositivos*. Luego, el descriptor *features.context* es recuperado para cada servicio contenido en el *Repositorio*. Posteriormente, la función verifica que el dispositivo sea compatible con las tecnologías de acceso definidas por el proveedor de servicios en el descriptor *features.context*.

2.3 Repositorio de servicios

Respaldado por el trabajo presentado en (Vanhatalo et ál., 2006), permite almacenar documentos BPEL y otro tipo de archivos basados en el estándar XML. Con estos archivos se enriquece el proceso de negocio con características de contexto. Para ello hemos definido un nuevo metadato que permite expresar/representar las restricciones o requerimientos del contexto del servicio.

El metadato denominado *features.context* contiene el nombre del servicio al cual está asociado (*Holidays*), una lista de los tipos de red soportados (*bluetooth, wifi, gsm, gprs*) y también una lista de la características de presentación/visualización que el dispositivo debe soportar para poder consumir los servicios invocados.

El proveedor de servicios publica tres tipos de documentos que describen sus servicios: BPEL, WSDL y *features.context*, archivos almacenados en el repositorio de servicios. Cabe resaltar que múltiples características se pueden definir en el metadato *features.context*; sin embargo, para el presente artículo se consideró solo el tipo de acceso como característica de contexto (*networkaccesses*), ya que el estudio y valoración de un solo parámetro permite establecer su funcionalidad. No obstante, la posibilidad de tener en cuenta más parámetros está abierta, tal como se presenta en la Figura 3.

Figura 3. Metadato de los requerimientos de contexto del servicio

```
<?xml version="1.0" encoding="UTF-8"?>
<emfcontext xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:emfcontext="http:///com/ibm/bpia/repository/extensions/emfcontext.ecore" name="Holidays">
 <networkaccesses>
        <networkaccess>bluetooth</networkaccess>
        <networkaccess>wifi</networkaccess>
        <networkaccess>gsm</networkaccess>
        <networkaccess>gprs</networkaccess>
 </networkaccesses>
 <display>
<resolution_width>320</resolution_width>
<resolution_height>480</resolution_height>
<columns>11</columns>
<max_image_width>320</max_image_width>
<max_image_height>36</max_image_height>
<rows>6</rows>
 </display>
</emfcontext>
```

Fuente: presentación propia de los autores.

2.4 Repositorio de dispositivos

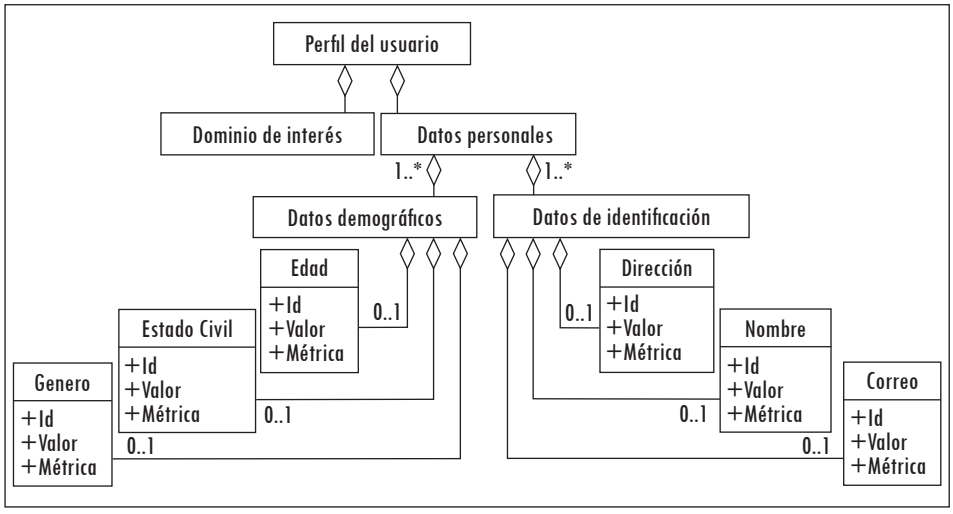
Soporta la segunda variable que corresponde a la descripción de capacidades de los dispositivos que solicitan los servicios, denominada *restricciones del contexto del solicitante*. Para esto se consideraron las siguientes características de contexto:

- *Espacial*: contiene todos los parámetros asociados con información geográfica y espacial, como son la longitud y latitud desde la cual un usuario accede al servicio.
- *Temporal*: contiene información sobre el momento (tiempo) y la fecha en la cual se hace una petición al sistema o es invocado el servicio.
- *Datos del equipo*: representa la información del equipo como *software* soportado e instalado. *Hardware* y tipo de sistema operativo son útiles para procesos de

adaptación de contenido. Estos parámetros abarcan, además, la información contenida en diferentes estándares utilizados para este propósito como el CC/PP. 3.5 Repositorio de Usuarios.

De Guerrero et ál. (2010) se tiene que los diferentes conjuntos de datos que conforman un *Perfil de Usuario* y que se han denominado *Dimensiones del Perfil de Usuario* son: datos personales y dominio de intereses. En la Figura 4 se presenta el metamodelo general del *Perfil de Usuario* definido. A continuación se explican cada una de estas dimensiones.

Figura 4. Metamodelo del perfil de usuario



Fuente: presentación propia de los autores.

- *Dominio de interés*: la información de esta dimensión, obtenida de un usuario, depende del ámbito de aplicación. La definición de estos parámetros y valores no se establece en este trabajo, debido al alto nivel de análisis y la desvinculación a un ámbito específico.
- *Datos personales*: la dimensión de los datos personales se divide en dos categorías (la de los datos de identificación y los datos demográficos). Esto se establece, debido a los requerimientos de protección de la identificación del usuario.

Estas descripciones se almacenan en el *Repositorio de Usuarios*. A partir de esta el sistema de descubrimiento realiza un proceso de sugerencia de servicios. La finalidad de este proceso es recomendar un grupo de servicios ya utilizados por

el usuario o personas con perfiles similares, y así disminuir el tiempo que toma el descubrimiento realizado sobre todo el *Repositorio de Servicios*.

El primer paso en la fase de sugerencia es encontrar usuarios con perfiles similares, para después utilizar los servicios consultados y consumidos por ellos. La función *FindSimilarProfiles* es la encargada de organizar una búsqueda sobre los perfiles contenidos en el *Repositorio de Usuario* (Algoritmo 3). Los perfiles de usuario están definidos de acuerdo con el metamodelo de *Perfil de Usuario* presentado. Para determinar la distancia entre los perfiles, la función *FindSimilarProfiles* usa la función *BasicProfileMatch*.

Algoritmo 3. Función *FindSimilarProfiles*

```
INPUTS: (queryProfile)
OUTPUT: similarProfilesList
BEGIN
List Profiles existingProfiles = lookupServiceRepository(non-operational information)
for each profile_i in existingProfiles do
Distance Profile distance = basicProfileMatch(profile_i, queryProfile)
if distance < 1 then
Add (profile_i, distance) to similarProfilesList
end if
end for
return similarProfilesList.
END
```

La función *basicProfileMatch* (Algoritmo 4) calcula la distancia entre una pareja de perfiles. Este cálculo se realiza sobre valores de características que describen el *Perfil de Usuario*. Debido al aumento en procesamiento y tiempo, se descartó la posibilidad de utilizar razonadores en esta comparación. Por eso la distancia es calculada utilizando un diccionario léxico como WordNet, que hace este procedimiento similar al emparejamiento de actividades básicas. Esta función comienza evaluando la similitud de las edades *SimAge,* luego evalúa la similitud entre el estado civil *SimCS*y la similitud de género *SimG*; continúa con el cálculo de la similitud de valores de la lista de dominios de interés $SimDoI_{ij}$ para estimar la similitud *SimDoI* general. Por último se calcula distancia entre los dos perfiles, *DistanceProfile*. Los pesos W_{age}, W_{cs}, W_{g} y W_{DoI} indican la contribución de la similitud de los atributos $DoIlist_i$, Age_i, CS_i, y G_i a la similitud de los perfiles ($0 \leq W_{age} \leq 1$, $0 \leq W_{cs} \leq 1$, $0 \leq W_{g} \leq 1$ y $0 \leq W_{DoI} \leq 1$). Para calcular la similitud de los valores de *DoI* se emplea la función *LinguisticSimilarity*(*LS*).

Algoritmo 4. Función *BasicProfileMatch*

INPUTS (Profile$_i$, Profile$_j$) Profile$_i$: Struct ($DoIlist_i$, Age_i, CS_i, G_i), Profile$_j$: Struct ($DoIlist_j$, Age_j, CS_j, G_j)
OUTPUT: DistanceProfile
BEGIN
Calculate Age Similarity $SimAge = 1-\{[|Age_i - Age_j|/[(Age_i+Age_j)/2]\}$
SimCS = 0, SimG = 0, SimDoI = 0
if $CS_i = CS_j$ **then**
SimCS =1
end if
if $G_i = G_j$ **then**
$SimG$ =1
end if
for each value v_i **in** $doilist_i$ **do**
for each value v_j **in** $doilist_j$ **do**
Calculate DoI$_{ij}$ Similarity $SimDoI_{ij} = LS(v_i,v_j)$
If $SimDoI_{ij} > SimDoI$ **then**
$SimDoI = SimDoI_{ij}$
end if
end for
end for

$$\text{DistanceProfile} = 1 - \frac{w_{age}*\text{SimAge} + w_{cs}*\text{SimCS} + w_g*\text{SimG} + w_{DoI}*\text{SimDoI}}{w_{age} + w_{cs} + w_g + w_{DoI}}$$

return DistanceProfile
END

En el Algoritmo 5 se expone el procedimiento para conseguir los servicios sugeridos antes del refinamiento de la búsqueda por parte de la función *Check-DeliveryContext*. En este se observan dos funciones que evidencian la utilidad del *Perfil de Usuario* en el proceso de sugerencia de servicios. Estas son: *listRanked-ServicesQueried* y *consumedNodes*. Internamente estas dos funciones utilizan a la función *FindSimilarProfiles*.

Algoritmo 5. Servicios sugeridos (*SuggestServices*)

```
INPUTS: (queryNode, usrProfile)
OUTPUT: listRankedServices
BEGIN
if searchServicesPreviouslyConsulted(queryNode) then
Get listRankedServices = listRankedServicesQueried(queryNode)
else
Get List NodessuggestServices = consumedNodes(usrProfile)
for each node_i in suggestService do
CalculteDistanceNodedistance = basicActivityMatch(node_i, queryNode)
if distance < 1 then
Add (node_i, distance) to listRankedServices order by distance
end if
end for
end if
if badSuggest(listRankedServices) then
listRankedServices = null
Get List Nodes candidateService = lookupServiceRepository(non-operational information)
for each node_i in candidateService do
Calculate Distance Node distance = basicActivityMatch(node_i, queryNode)
if distance < 1 then
Add ( node_i, distance) to listRankedServices order by distance
end if
end for
end if
Return listRankedServices
END
```

La función *SuggestServices* empieza verificando si el nodo de consulta ya ha sido solicitado previamente. En caso afirmativo, recupera del *Repositorio de Usuarios* la información de los servicios retornados en esa oportunidad. En este punto es importante distinguir la diferencia entre un servicio de consulta y un servicio sugerido. El servicio de consulta es entregado por el usuario, contiene una descripción de sus requerimientos y puede ser un servicio abstracto o un servicio de ejecución. En cambio, los servicios sugeridos, obligatoriamente, deben poseer una implementación (servicios de ejecución).

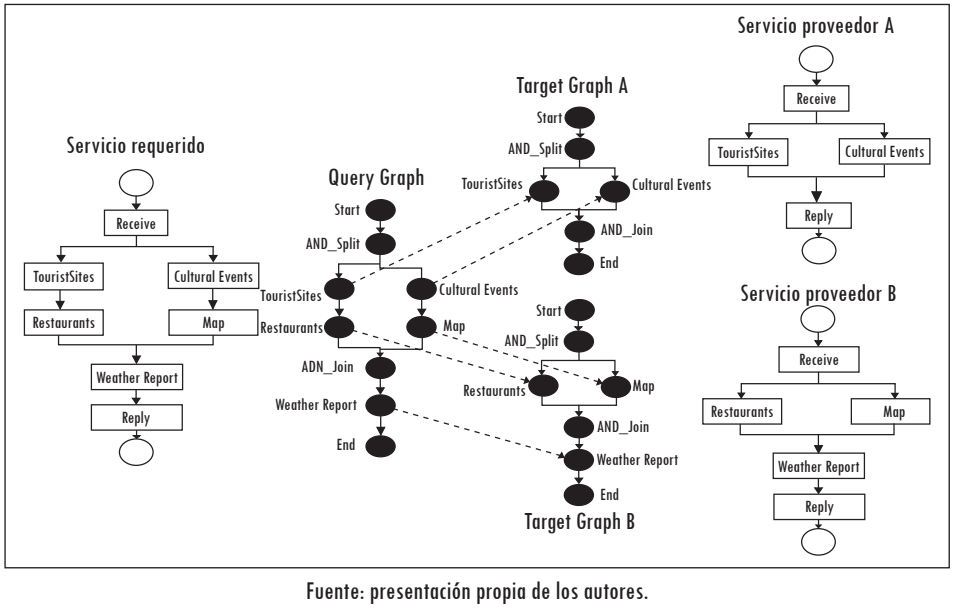
La importancia de los servicios de consulta radica en que si bien no poseen una implementación real (servicio abstracto), tienen relacionada una lista de servicios reales que les fueron sugeridos. Pero si el servicio de consulta no ha sido solicitado en ocasiones anteriores, se realiza una primera búsqueda sobre aquellos nodos que han sido consumidos por el usuario o usuarios con perfiles similares. Estos nodos son organizados basándose en su distancia semántica al nodo de consulta *queryNode*. Una vez terminada esta organización, la función *badSuggest* verifica

que los nodos de *listRankedServices* cumplan con unas condiciones (superando un umbral y un número de servicios mínimos requeridos por el usuario) para ser entregados al usuario como los más pertinentes a sus requerimientos. Si esta lista no cumple con dichas condiciones, la búsqueda comienza desde cero analizando todo el *Repositorio de Servicios*.

3. Ejemplo

En esta sección se ejemplifica el mecanismo propuesto de descubrimiento de servicios para ambientes de computación ubicua. Supongamos que un turista requiere un servicio de información turística que relacione sitios turísticos, eventos culturales y restaurantes en un mapa de la ciudad. Adicionalmente, el turista solicita el reporte del clima para programar su *tour*. El turista proporciona este requisito a través de su dispositivo móvil, el cual es convertido en un documento BPEL. Luego la plataforma toma este archivo y lo transforma en su equivalente en grafos. Para este caso el servicio solicitado tiene el siguiente modelo: *TouristSites* (tipo: *Invoke*), *CulturalEvents* (tipo: *Invoke*), *Restaurants* (tipo: *Invoke*), *Map* (tipo: *Invoke*) y *WeatherReport* (tipo: *Invoke*). El usuario envía esta descripción BPEL a la plataforma utilizando SOAP/HTTP con el fin de obtener los servicios más apropiados (Figura 5).

Figura 5. Ejemplo



Fuente: presentación propia de los autores.

El *Repositorio de Servicios* contiene un número considerable de procesos BPEL publicados. Por esta razón, a fin de obtener los requerimientos del turista, nuestra plataforma ejecuta el Algoritmo 5, el cual considera tanto el *perfil del usuario* como el nodo de consulta (*queryNode*).

Ahora supondremos que del proceso anterior se recuperaron los siguientes servicios, publicados por dos proveedores de servicios. El proveedor de servicios A publicó los siguientes servicios: *TouristSites* (*invoke*) y *CulturalEvents* (*invoke*). El proveedor de servicios B publicó los siguientes servicios: *Maps* (*invoke*), *Restaurants* (*invoke*) y *WeatherReport* (invoke).

Cabe resaltar que previamente la plataforma convirtió cada documento BPEL en su equivalente grafo (*QueryGraph*, *Target GraphA* y*Target Graph B*, Figura 5). Los grafos presentados en la Figura 5 tienen algunas actividades en común, a pesar de tener una estructura diferente. El algoritmo de emparejamiento propuesto encuentra las actividades semánticamente más similares, independientemente de la estructura de los grafos (*Query* y *Target*). Así, las líneas punteadas en la Figura 6 representan las coincidencias encontradas por el sistema entre los grafos usando la función *BasicActivityMatch.*

Una vez recuperados los servicios, estos son filtrados de acuerdo con el contexto de entrega del solicitante. Determinar el contexto de entrega implica encontrar el perfil del dispositivo utilizado por el usuario (por ejemplo, el turista utiliza el dispositivo móvil Nokia 6310i). El perfil del dispositivo se obtiene de los repositorios UAProf y WURFL. La URL del documento UAProf de un dispositivo móvil se encuentran frecuentemente en las cabeceras HTTP. La Figura 6 muestra el documento UAProf para el Nokia 6310i.

Figura 6. UAProf Nokia 6310i

```
<prf:component>
  <rdf:Description rdf:ID="NetworkCharacteristics">
    <rdf:type rdf:resource="http://www.wapforum.org/profiles/UAPROF/ccppschema-
20010430#NetworkCharacteristics"/>
      <prf:SupportedBluetoothVersion>1.1</prf:SupportedBluetoothVersion>
      <prf:SecuritySupport>
        <rdf:Bag>
          <rdf:li>WTLS-1</rdf:li>
          <rdf:li>WTLS-2</rdf:li>
        </rdf:Bag>
      </prf:SecuritySupport>
      <prf:SupportedBearers>
        <rdf:Bag>
          <rdf:li>CSD</rdf:li>
          <rdf:li>GPRS</rdf:li>
          <rdf:li>GSM900</rdf:li>
          <rdf:li>GSM1800</rdf:li>
          <rdf:li>GSM1900</rdf:li>
        </rdf:Bag>
      </prf:SupportedBearers>
  </rdf:Description>
</prf:component>
```

Fuente: presentación propia de los autores.

Las características de red en el documento UAProf se utilizan para comprobar que el dispositivo soporta el tipo de acceso definido por el proveedor de servicios en el documento *features.context*. Esta comprobación es realizada por la función *CheckDeliveryContext*, descrita en el Algoritmo 2. Finalmente, una lista ordenada de los servicios más similares que pueden ser consumidos desde el terminal móvil donde se efectuó la consulta se retorna al turista.

4. Experimentación

Para evaluar la eficacia del algoritmo de recuperación propuesto se utilizó una herramienta de evaluación de técnicas de descubrimiento de servicios basada en una metodología de *benchmarking,* que analiza a partir de la comparación de evaluaciones realizadas a dos o más técnicas o sistemas que tengan una base común de entradas. A estas evaluaciones se las denomina *benchmark del algoritmo* y *benchmark de referencia*. El *benchmarking* es el proceso encargado de comparar los dos *benchmarks.* Así, para la evaluación se procedió de la siguiente manera: 1) el *benchmark de referencia* se construyó con el propósito de tener una base, para

comparar un conjunto de actividades de consulta contra un repositorio de servicios; esto constituye una "verdad absoluta" sobre la similitud de las actividades comparadas. 2) La construcción del *benchmark de referencia* se da paso a la recolección de datos para generar un *benchmark del algoritmo.* Y 3) Se compararon los resultados arrojados por el algoritmo de recuperación contra el *benchmark de referencia*, con el objeto de medir su desempeño.

El *benchmark de referencia* se creó comparando 30 actividades básicas BPEL contra 144 actividades almacenadas en el repositorio de servicios, y así se obtuvieron 1.106 parejas para evaluar. Las evaluaciones fueron realizadas por cinco evaluadores expertos en el tema, para un total de 5.530 comparaciones. Las comparaciones realizadas para las actividades BPEL evalúan la similitud de cinco atributos: *ActivityType, Operation, Portype, PartnerLink* y *AccessType*. El evaluador comparó entre las actividades asignando una calificación a cada uno de los atributos según su similitud. El valor de la calificación está entre 0 y 5 (0 es mínima similitud y 5 es máxima similitud). El experto evaluador fija un peso de acuerdo con la importancia de cada uno de los atributos según su criterio. La suma total de los pesos debe ser igual a uno.

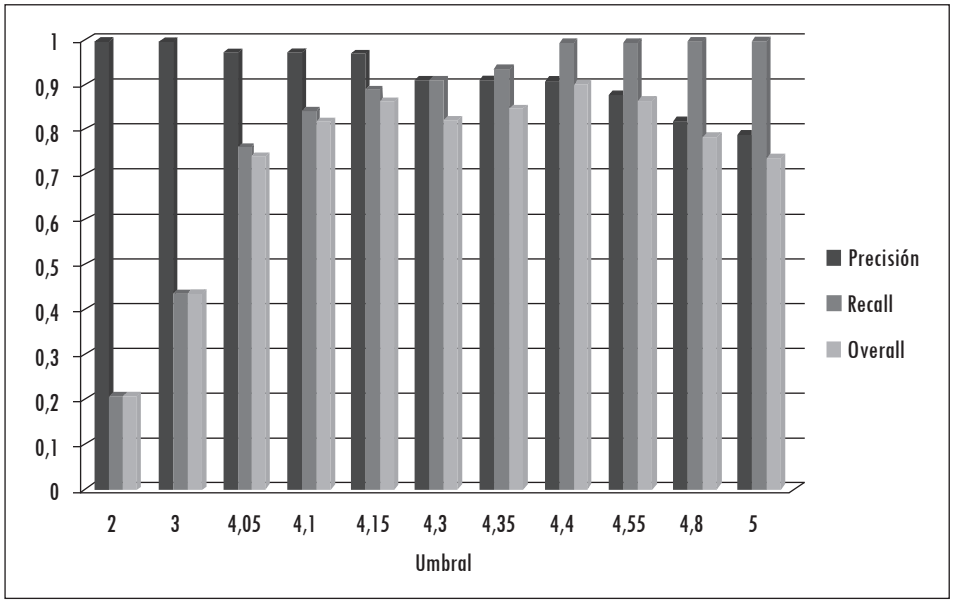
El sistema de recuperación de servicios trabaja sobre un repositorio de procesos BPEL (Vanhatalo et ál., 2006), el cual almacena 144 actividades básicas BPEL. El algoritmo de emparejamiento se implementó en lenguaje Java y los experimentos fueron realizados en un computador con procesador Pentium 4 de 2,30 GHz, 1.000 MB de RAM, bajo el SO Linux Ubuntu.

La herramienta de *bencharking* empleada permite realizar un análisis estadístico, mediante estrictas medidas usadas comúnmente por diversas investigaciones enfocadas en la recuperación de información, como *Precisión, Recall, Overall, Top-k Precisión* y *P-Precisión*. Los valores globales de estas medidas se calcularon por medio de la técnica de micropromedio descrita en (Lewis, 1992).

Así, del análisis obtenido por la herramienta de *benchmarking*, en la Figura 7 se puede apreciar el desempeño total del sistema, donde se identifica el umbral de similitud óptimo para el algoritmo de recuperación, equivalente a 4,4, punto en cual la medida *Overall* alcanza su tope en un valor cercano al 90%. Igualmente, se aprecia que en los valores de similitud superiores a 4,0 el sistema obtiene un mejor desempeño (manteniendo el *Overall* sobre 70% para valores de similitud entre 4 y 5) que aquellos ubicados en la parte inferior. De ahí que sea más favorable trabajar con valores altos de similitud, donde la medida de *Recall* realiza un mayor aporte al desempeño general. El algoritmo trabaja de una manera muy selectiva, tomando valores de precisión muy altos y recuperando

un alto porcentaje de servicios relevantes cuando el umbral de similitud está por encima de 4.

Figura 7. Desempeño del sistema



Fuente: presentación propia de los autores.

5. Conclusiones

En este artículo se propone una plataforma para recuperar servicios en ambientes de computación ubicua basada en personalización. Igualmente, se resalta la importancia de considerar dentro del proceso de descubrimiento una etapa de emparejamiento atómica que opere sobre modelos de comportamiento; en este caso BPEL, que permite evaluar la distancia semántica entre los servicios presentes en la red ubicua y los requeridos por el usuario, mediante una comparación lingüística de los parámetros de las actividades básicas BPEL.

La personalización del proceso de descubrimiento de servicios se aborda desde la gestión del contexto y el perfil de usuario, que evidencia la utilidad de los repositorios de usuarios, dispositivos y servicios, como herramientas de soporte al proceso de descubrimiento de servicios en entornos tan dinámicos y heterogéneos como los ambientes de computación ubicua.

Para evaluar la eficacia del algoritmo de descubrimiento de servicios propuesto se empleó una herramienta de *benchmarking*, con el fin de garantizar una

evaluación objetiva y confiable. Se determinó el umbral de similitud óptimo, equivalente a 4,4, valor en el cual se alcanza el máximo desempeño del sistema de recuperación. Se evidenció, además, que el desempeño es mejor cuando se emplean valores de similitud superiores a 4, ya que en valores bajos de similitud la medida de *recall* es muy pobre al descartar demasiadas actividades consideradas como relevantes.

Referencias

ABBAR, S.; BOUZEGHOUB, M.; KOSTADINOV, D.; LOPES, S.; AGHASARYAN, A. y BETGE-BREZETZ, S. A personalized access model: concepts and services for content delivery platforms. En: *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*. Linz: ACM, 2008.

ALMENÁREZ, F. *Arquitectura de seguridad para entornos de computación ubicua abiertos y dinámicos*. Tesis doctoral. Escuela Politécnica Superior, 2005.

ANDREWS, T. et ál. *Business Process Execution Language Version 1.1. The BPEL4WS Specification*. [web en línea]. <http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf> [Consulta: 10-08-2010].

ANGELL, R. C.; FREUND, G. y WILLETT, P. Automatic spelling correction using a trigram similarity measure. *Information Processing & Management*. 2002. vol. 19, núm. 4, pp. 255-261.

BANDARA, A.; TERRY, P.; ROURE, D. D. y TIM, L. *A semantic approach for service matching in pervasive environments*. Hampshire: Universidad de Southampton, 2007.

BELOTTI, R; DECURTINS, C.; GROSSNIKLAUS, M.; NORRIE, M. y PALINGINIS, A. Modelling context for information environments. *Ubiquitous Mobile Information and Collaboration Systems*. 2005, vol. 3272, pp. 43-56.

BEN MOKHTAR, S.; GEORGANTAS, N. e ISSARNY, V. COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support. *Journal of Systems and Software*. 2007, vol. 80, núm. 12, pp. 1941-1955.

CORRALES, J. C.; GRIGORI, D.; BOUZEGHOUB, M. y GATER, A. Ranking. BPEL Processes for Service Discovery. IEEE Transactions on Services Computing. 2010, vol. 3, núm. 3, pp. 178-192.

CORRALES, J. C. *Behavioral matchmaking for service retrieval*. PhD thesis. Versailles: University of Versailles Saint-Quentin-en-Yvelines, 2008.

GUERRERO, E.; CORRALES, J. C. y RUGGIA, R. *Selección de servicios basado en metamodelos del perfil, contexto y QoS*. s. l.: EATIS, 2010.

LEWIS, D. *Representation and learning in information retrieval*. Doctoral Thesis. Boston: Department of Computer and Information Science, University of Massachusetts, 1992.

MILLER, G. Wordnet: A lexical database for english. *Communications of the ACM*. 1995, vol. 38, núm. 11, pp. 39-41.

SELLAMI, M.; TATA, S. y DEFUDE, B. *Service discovery in ubiquitous environments. Approaches and requirement for context-awareness*. Milan: BPM Workshops, 2009.

STELLER, L. y KRISHNASWAMY, S. Efficient mobile reasoning for pervasive discovery. *Proceedings of the 2009 ACM symposium on Applied Computing (SAC)*. 2009, pp. 1247-1251.

VANHATALO, J.; KOEHLER, J. y LEYMANN, F. Repository for business processes and arbitrary associated metadata. *Proceedings of the BPM Demo Session at the Fourth International Conference on Business Process Management (BPM 2006)*. Vienna, Austria, 5-7 septiembre 2006, pp. 25-31.

WEISER, M. The computer for the 21st century. *Scientific American*. 1991, vol. 265, núm. 3, pp. 94-104.