COMPOSICIÓN DE SERVICIOS EN AMBIENTES DE COMPUTACIÓN UBICUA

Victor A. Hermida
Grupo de Ingeniería Telemática (GIT) – Universidad del Cauca
Calle 5 No 4-70 - Popayán, Colombia
(57-2) 8 369033
vhermida@unicauca.edu.co

Luis Javier Suarez
Grupo de Ingeniería Telemática (GIT) – Universidad del Cauca
Calle 5 No 4-70 - Popayán, Colombia
(57-2) 8 369033
ljsuarez@unicauca.edu.co

Luis Antonio Rojas
Grupo de Ingeniería Telemática (GIT) – Universidad del Cauca
Calle 5 No 4-70 - Popayán, Colombia
(57-2) 8 369033
luisrojas@unicauca.edu.co

Juan Carlos Corrales
Grupo de Ingeniería Telemática (GIT) – Universidad del Cauca
Calle 5 No 4-70 - Popayán, Colombia
(57-2) 8 369033
jcorral@unicauca.edu.co

Oscar Caicedo
Grupo de Ingeniería Telemática (GIT) – Universidad del Cauca
Calle 5 No 4-70 - Popayán, Colombia
(57-2) 8 369033
omcaicedo@unicauca.edu.co

ABSTRACT
Ubiquitous computing has established itself as the next generation of communications, offering to users the capabilities of the networks at any time and place, in order to support their daily tasks. However, the deployment of services for ubiquitous computing environments requires several challenges, such as mobility, dynamism and heterogeneity. Generally, providing a service to users is relatively simple, but the demand for these is becoming more demanding, thereby inducing an innovative feature and therefore a high complexity and variability of services. To comply with the requirements of users, it is necessary to combine several services, which implies provide to ubiquitous environments of efficient mechanisms of discovery and composition. In this paper, we propose a service composition platform for ubiquitous computing environments, which considers the specification of the behavior of services, the user profile and context. Thus, it is possible to obtain a composite service, which apart from meeting the demand of the user, adjusts to your preferences and the context in which it is.

RESUMEN
La computación ubicua se ha establecido como la siguiente generación de las comunicaciones, ofreciendo a los usuario las capacidades de la redes en cualquier momento y lugar, con el fin de soportar sus tareas cotidianas. Sin embargo, el despliegue de servicios para los ambientes de computación ubicua requiere afrontar diversos retos, tales como la movilidad, el dinamismo y la heterogeneidad. Generalmente, ofrecer un servicio al usuario es relativamente sencillo, pero la demanda de éstos es cada vez más exigente, induciendo así un carácter innovador y por ende una alta complejidad y variabilidad a los servicios. Con el fin de dar cumplimiento a las exigencias de los usuarios, es necesario combinar varios servicios, lo cual implica dotar a los ambientes ubicuos de mecanismos eficientes de descubrimiento y composición. En el presente trabajo, se propone una plataforma de composición de servicios para ambientes de computación ubicua, que considera la especificación del comportamiento de los servicios, así como el perfil y contexto del usuario. De esta forma, es posible obtener un servicio compuesto, que además de satisfacer la demanda del usuario, se ajusta a sus preferencias y al contexto en el cual se encuentra.

Categories and Subject Descriptors
H.3.5 [**Information System**]: Online Information Services, Commercial services, Data sharing, Web-based services.

General Terms
Algorithms, Performance, Design, Experimentation, Languages.

Keywords
Ambiente ubicuos, descubrimiento de servicios, composición de servicios, perfil de usuario, contexto.

1. INTRODUCCIÓN
La computación ubicua puede ser descrita como dispositivos portables con facilidades embebidas de computación y comunicación, cuya meta es copar el ambiente con componentes electrónicos para ayudar de una manera natural a los usuarios durante sus tareas diarias [2]. Para lograr los objetivos de la computación ubicua, se requiere abordar retos significativos, principalmente relacionados con la heterogeneidad del ambiente, el dinamismo y la información que describe al usuario (perfil y contexto) [3, 4]. La computación ubicua debe soportar las tareas de los clientes por medio de la integración de las funcionalidades de los servicios de red, de tal forma, que éstos puedan ser invocados en cualquier momento, lugar y dispositivo de acceso [2]. Supongamos que un usuario necesita: i) un servicio de información turística que ofrezca información relacionada con los sitios de interés y restaurantes ubicados en la ciudad objeto de su

visita. ii) Un reporte del clima para poder programar su recorrido en la ciudad. En este escenario, el turista realiza su petición por medio de una aplicación móvil y el ambiente de computación ubicua buscará los servicios de información sobre sitios turísticos, restaurantes y predicción del clima para poder crear un servicio compuesto que responda a sus exigencias.

Sin embargo, encontrar un servicio que cumpla exactamente con las exigencias de búsqueda del usuario se convierte en un caso excepcional, ya que la mayoría requieren de una adaptación de los servicios disponibles en la red o un proceso de composición de diversas capacidades para ejecutar tareas complejas [5, 6]. Según la literatura, el mecanismo para combinar dos o más servicios disponibles y formar un nuevo servicio se denomina *Composición de servicios*. Esta composición puede ser estática o dinámica. Así, la primera es realizada durante la fase de despliegue, sin embargo, esta solución limita el posible uso de los recursos y no se adapta a las fluctuaciones propias de los ambientes ubicuos [7]. Mientras que la composición dinámica, busca combinar los servicios ubicuos disponibles en la red durante la fase de ejecución, brindando una mejor respuesta a los requisitos de heterogeneidad y dinamismo de los ambientes de computación ubicua [8-10].

De este modo, en contextos con este tipo de características, es imprescindible contar con capacidades de descubrimiento y composición de servicios, que permitan explotar los recursos (aplicaciones, archivos, almacenamiento, hardware, etc.) presentes en la red. El descubrimiento y composición de servicios deben incorporar las características necesarias para ser tan dinámicos y autónomos como las condiciones de la red ubicua lo requieran.

En el presente trabajo se propone un mecanismo de descubrimiento y composición de servicios. La fase de descubrimiento se trata en extenso en [1]. Los algoritmos utilizados en dicha fase actúan sobre servicios descritos con BPEL (Business Process Execution Language) [11]. Asimismo, las técnicas de descubrimiento consideradas, operan sobre información relacionada con el usuario, tal como su perfil y el contexto, con el objetivo de recuperar servicios que se ajusten a sus preferencias y puedan ser consumidos por el dispositivo móvil de acceso. El presente artículo, describe principalmente la etapa de composición de servicios, la cual se encarga de componer el proceso más similar al servicio requerido por el cliente, usando los servicios recuperados en la etapa de descubrimiento. Las técnicas utilizadas en esta fase, identifican de manera única las actividades involucradas en la composición. Posteriormente, analizan la conformidad de los servicios a componer, a través de la construcción de un grafo de dependencia de datos que permite establecer la compatibilidad de los servicios a partir de las interfaces. Luego, se obtiene la síntesis de composición, la cual esta soportada en el grafo de dependencia de datos, y el plan de la composición requerido.

En la siguiente sección, se presentan los trabajos relacionados. La sección 3, explica la arquitectura propuesta para la plataforma de descubrimiento y composición automática de servicios en ambientes de computación ubicua, denominada U-BeComp. En la sección 4 y 5 se describen las fases de descubrimiento y composición respectivamente. En la sección 6 se analizan los resultados obtenidos después de evaluar experimentalmente la plataforma. Por último, en la sección 7 se presentan las conclusiones y trabajos futuros.

2. TRABAJOS RELACIONADOS

La composición de servicios se puede dividir en dos aspectos fundamentales, la síntesis y la orquestación [8, 12]. La síntesis se refiere a cómo generar un plan para lograr el comportamiento deseado, a través de la combinación de múltiples servicios. La orquestación se encarga de la coordinación del flujo de control y datos entre varios componentes durante la ejecución del plan. El presente artículo se enfoca exclusivamente en la síntesis autónoma de servicios, la orquestación de los servicios es un problema complementario que será abordado en trabajos futuros.

En [3] se propone una clasificación de soluciones para la composición de servicios basada en dos categorías: la composición basada en interfaces y la composición basada en conversaciones. La composición basada en interfaces es empleada cuando los servicios disponibles en la red y las tareas de usuario, son descritos como capacidades individuales sin una conversación asociada; en este modelo, los servicios son combinados basándose en la conformidad de sus signaturas [3]. En esta categoría se pueden encontrar enfoques basados en búsquedas sobre grafos [7, 13] y soluciones soportadas en el encadenamiento de servicios [14-17]. En el grafo construido en [2] los nodos representan los servicios disponibles y las aristas indican la correspondencia entre las salidas de un servicio y las entradas a otro. El problema de esta solución radica en su pobre escalabilidad cuando se incrementa el número de servicios contenidos en los repositorios, lo cual se ve explícito en su complejidad $O(n^{3)}$. El trabajo de [7] presenta una aproximación para la agregación de servicios basada en grafos que permite la composición autónoma de servicios en ambientes ubicuos, construyendo un grafo de dos niveles, el primero representa las relaciones funcionales de los servicios y sus parámetros, mientras el segundo contiene las instancias de los servicios y los tipos de datos. La complejidad está determinada por $O(m^2)$, donde *m* representa una medida del tamaño de la ontología empleada para las comparaciones semánticas.

La composición de servicios basada en conversaciones, asume que los servicios poseen un comportamiento complejo, descrito a través de una conversación [8]. La selección de conversaciones dirigida por objetivos, busca un comportamiento que responda a la tarea que el usuario requiere, en este caso no existe una integración con servicios que pertenezcan a otros procesos [19]. La integración de tareas dirigida por objetivos, selecciona un conjunto de conversaciones y las integra de tal forma que el servicio compuesto satisface la tarea de usuario, consumiendo las entradas proporcionadas y generando los efectos requeridos. En este enfoque, las tareas de usuario se expresan en términos de sus entradas y salidas, buscando una mayor facilidad en la formulación de los requisitos del usuario [10, 20]. La integración de conversaciones dirigida por conversaciones, captura el comportamiento en la tarea de usuario, y además, asume que tanto los servicios como las tareas de usuario son expresados por medio de una conversación [3, 9, 13], los servicios compuestos se generan de acuerdo a la conversación definida en las tareas de usuario.

En [9] los autores presentan el framework Colombo, el cual aborda el problema de la composición automática de servicios web. Este trabajo incorpora la noción de “servicio meta”, para denotar el comportamiento deseado para el servicio requerido. En [13] los servicios y los requisitos de los usuarios son expresados como una terna ordenada compuesta por un conjunto de entradas, un conjunto de salidas y un conjunto de dependencias entre las entradas y las salidas. El conjunto de dependencias expresan el

procesamiento que se desea para manejar los datos entregados por el usuario, este procesamiento puede ser realizado por uno o varios servicios atómicos. La composición es realizada por medio de la búsqueda en un grafo de dependencia. En [3] se presenta un mediador para la composición de servicios para ambientes ubicuos, en este trabajo las tareas de usuario son expresadas como una dupla conformada por un autómata que describe la conversación de la tarea y un conjunto de propiedades no funcionales que deben cumplir los servicios. La composición esta soportada en una representación de autómatas finitos y busca la integración de servicios, recuperados en una fase previa de emparejamiento semántico. La complejidad computacional de la solución está determinada por la función $O(m^{nb})$, donde *nb* es la cantidad de capacidades requeridas en la conversación y *m* es el número de capacidades contra el cual se compara la solicitud del usuario.

De las técnicas de composición expuestas, se tiene que el enfoque basado en conversaciones posee una mayor confiabilidad, al asegurar que interven los servicios requeridos por el usuario, dando así claridad sobre los datos que debe manipular cada uno de ellos, obteniendo un flujo de ejecución y de control similar al deseado. Por otro lado, la complejidad computacional de los anteriores trabajos no es la más adecuada en entornos tan dinámicos y heterogéneos como los ambientes de computación ubicua. De acuerdo a esto, el presente trabajo emplea una técnica de composición dirigida por conversaciones, que busca dar solución a la complejidad inherente a la composición de servicios. Así, para garantizar una mejor escalabilidad, las conversaciones solicitadas están compuestas por servicios atómicos, recuperados en la fase previa de descubrimiento, donde se encuentran los servicios más similares a los requeridos por el usuario.

3. ARQUITECTURA

Esta sección presenta la arquitectura de la plataforma de descubrimiento y composición automática de servicios para ambientes de computación ubicua, denominada U-BeComp. En el enfoque propuesto, tanto los servicios disponibles en la red ubicua, como las solicitudes del usuario son descritos mediante BPEL. La Figura 1 describe brevemente U-BeComp y sus módulos.

El usuario interactúa con la plataforma por medio de una aplicación, la cual soporta el envío de peticiones expresadas como modelos de comportamiento BPEL. El sistema recibe las solicitudes y las transforma a una representación formal de grafos, para posteriormente ejecutar el descubrimiento y la composición de servicios. El descubrimiento de servicios considera las preferencias de los usuarios móviles, las capacidades de los dispositivos de acceso y el contexto de entrega, con el objetivo de proveer flexibilidad para reconfigurar los servicios de acuerdo a los cambios del entorno. La composición está encargada de componer los servicios recuperados por el módulo de descubrimiento, con el fin de crear una conversación útil que ejecute las tareas de usuario establecidas en el requerimiento inicial.

Finalmente, la plataforma retorna al usuario el servicio compuesto, para que los servicios atómicos que lo conforman sean invocados por medio del motor BPEL. 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. Los módulos que componen a U-BeComp son detallados a continuación:

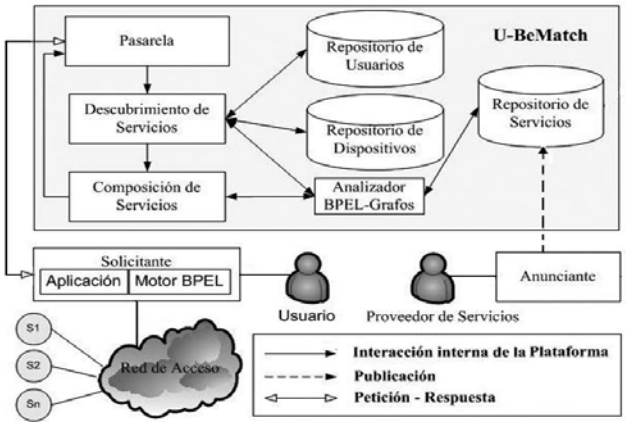

Figura 1. Arquitectura U-BeComp

•**Solicitante**: módulo compuesto por una aplicación móvil y un motor BPEL. La aplicación ofrece la interfaz gráfica para la interacción con el usuario, además de proveer las capacidades de comunicación para el envío y recepción de los procesos BPEL. El motor BPEL es implementado por Sliver [21], un mediador que soporta la ejecución de procesos BPEL en un amplio rango de dispositivos, desde computadores de escritorio hasta dispositivos móviles.

•**Pasarela**: recibe las peticiones enviadas desde el Solicitante y transmite las repuestas generadas por los niveles superiores. Este componente brinda un acceso independiente de la red (Wifi, Bluetooth, etc.) y de los protocolos (HTTP, SMTP, SMS, etc.) a la plataforma.

•**Anunciante**: ofrece las capacidades necesarias para que los proveedores publiquen sus servicios en el repositorio de la plataforma. Las descripciones de los servicios contienen atributos funcionales y no-funcionales, tales como: entradas, salidas, precondiciones, restricciones, etc. Este módulo es implementado a través de una aplicación web.

•**Repositorios**: almacenan información sobre la descripción de los servicios disponibles, las preferencias y el contexto del usuario. Estos repositorios son: i) Repositorio de Servicios: utiliza el repositorio de procesos de negocio presentado en [22], el cual soporta el almacenamiento y consulta de documentos BPEL (y otros documentos XML). ii) Repositorio de Usuarios: almacena detalles relacionados con los usuarios, estos incluyen preferencias explicitas provistas por ellos (ej. Lenguaje nativo) o datos implícitos capturados dinámicamente tales como historial de servicios consumidos o patrones de uso y/o comportamiento. iii) Repositorio de dispositivos: almacena las características de los dispositivos tales como: capacidad de procesamiento, modalidades de presentación, interfaces de entrada, conectividad, etc. Esta información se recupera de los repositorios: UAProf (User Agent Profile) [23] y WURFL (Wireless Universal Resource) [24].

•**Analizador BPEL-Grafos**: transforma las descripciones de comportamiento (BPEL) en su equivalente en grafos, implementando la estrategia presentada por [25]. 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 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.

•**Módulo de Descubrimiento de Servicios**: recupera los servicios más apropiados para componer el modelo de comportamiento requerido por el usuario [1].

•**Módulo de Composición de Servicios**: encargado de componer el proceso más similar al servicio requerido por el cliente, usando los servicios recuperados en la etapa de descubrimiento. La composición se realiza teniendo en cuenta el flujo de control definido en la petición del usuario.

En las siguientes secciones, la fase de descubrimiento es descrita brevemente, mientras que el proceso de composición y las fases que lo conforman, como: la creación del grafo de dependencias y los procedimientos para realizar la síntesis de composición, son ampliamente detalladas.

4. FASE DE DESCUBRIMIENTO

En esta sección describiremos ligeramente el enfoque propuesto de la fase descubrimiento de servicios en ambientes de computación ubicua. El objetivo de esta etapa es proveer un mecanismo de recuperación de servicios, que tenga en cuenta tanto las preferencias del usuario, como las especificaciones de dispositivos y el contexto de entrega.

El Algoritmo 1 define la función para obtener un conjunto ordenado de servicios, los cuales cumplen con la solicitud del usuario. Este algoritmo hace uso de dos funciones: La función GetRankedServicesFromCache(n) provee una lista organizada de servicios desde la cache para cualquier solicitud n del usuario (si dichos servicios existen).

Si el usuario actual, u otros, no han enviado la solicitud n previamente, el algoritmo recupera una lista de servicios, los cuales, el usuario actual u otros usuarios con perfiles similares, han invocado anteriormente. La función ConsumedServices(pi) retorna dichos servicios, donde pi es el perfil del usuario actual.

El algoritmo obtendrá un servicio candidato, comparando el nodo de la solicitud del usuario ni contra cada servicio previamente invocado y lo adicionará a un lista organizada (ranking). CheckMatch(ni, np) es una función que retorna un double, indicando la similitud semántica / distancia entre el nodo solicitado ni y el nodo candidato np, para mayores detalles ver [1]. La función BadSuggest(RS) retorna verdadero si un ranking dado de servicios RS, no contiene suficientes servicios que cumplan con un determinado umbral de similitud en contraste a la solicitud del usuario. En caso tal de no encontrar un servicio que cumpla satisfactoriamente con el requerimiento del usuario (es decir, BadSuggest retorna true), entonces todos los otros servicios registrados en el Repositorio de Servicios serán comparados con el servicio solicitado con el fin de producir el ranking deseado. LookupServiceRepository denota la función que permite recuperar los servicios desde el Repositorio de Servicios. Posteriormente, la función checkDeliverycontext(RS) verifica si los servicios recuperados pueden ser invocados en el contexto del usuario. Finalmente, se retorna un conjunto de los servicios recuperados, ordenado según el valor de similitud estimado respecto a los servicios de consulta.

5. FASE DE COMPOSICIÓN

El proceso de composición se inicia creando un grafo de dependencias, el cual permite establecer la compatibilidad entre las interfaces de los servicios que deben interactuar en la realización de la tarea de usuario. El grafo de dependencia es construido empleando el concepto de Híper-grafos [26].

```
1.  INPUTS: Node n_q, UserProfile p
2.  OUTPUT: RankedList RS  /* ranked list of service nodes */
3.  BEGIN
4.  Let RS← GetRankedServicesFromCache(n_q)
5.  if RS != null then
6.     return RS
7.  else
8.     Let S ← ConsumedServices(p) /* where S is a set of nodes
       n_k, such that S = {n_1, …, n_p} */
9.     for each n_k in S do
10.       Let dist ← CheckMatch(n_k, n_q) /*see [1]*/
11.       if dist < 1 then
12.       RS ← RS ∪ (dist, n_k) /* add n_k to set RS, ordered by
                dist */
13.       end if
14.    end for
15. end if
16. if BadSuggest(RS) then
17.    RS ← null
18.    Let S = LookupServiceRepository(non-operational
       information)  /* where S is a set of nodes n_k, such that S =
       {n_1, …, n_p} */
19.    for each n_k in S do
20.       Let dist ← CheckMatch(n_k, n_q) /*see [1]*/
21.       if dist < 1 then
22.       RS ← RS ∪ (dist, n_k) /* add n_k to set RS, ordered by
                dist
23.       end if
24.    end for
25. end if
26. RS ← checkDeliverycontext(RS)
27. return RS
28. END
```

Algoritmo 1. Función de Recuperación de Servicios

5.1 Grafo de dependencias

El grafo de dependencia es construido empleando el concepto de Híper-grafos [26]. A continuación se definen formalmente los Híper-grafos.

Definición 1: *Híper-grafo*, es una dupla **H** = (**V**, **E**), donde **V** = $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$,…, $\mathbf{v_n}$ es el conjunto de vértices o nodos, y **E** = $\mathbf{E_1}$, $\mathbf{E_2}$, $\mathbf{E_3}$,…, **Em** con $\mathbf{E_i} \subseteq \mathbf{V}$ para **i** = 1, 2,3,…, m es el conjunto de híper-aristas. Cuando $|\mathbf{E_i}| = 2$, **i** = 1,2,…, **m** el híper-grafo es una grafo convencional.

Definición 2: *Híper-grafo dirigido*, es un híper-grafo con híper-aristas dirigidas. Una híper-arista dirigida es una pareja ordenada **E** = (**X**, **Y**), donde **X** es la cola de **E**, denotada por **T**(**E**), mientras **Y** representa la cabeza **H**(**E**).

Definición 3: *BF-Híper-grafo***,** un BF-híper-grafo, o simplemente *BF-grafo*, es un híper-grafo cuyas híper-aristas son del tipo **B-arista** o **F-arista**. Una B-arista (*backward*) es una híper-arista **E**= ((**T**(**E**),**H**(**E**)) con |**H**(**E**)|= 1). Una F-arista (*forward*) es una híper-arista **E**= ((**T**(**E**),**H**(**E**)) con |**T**(**E**)|= 1).

Para la construcción del grafo de dependencias, se emplean los nodos que representan las actividades recuperadas en la fase de descubrimiento y se crean nuevos nodos para representar los datos que conforman los mensajes de entrada y/o salidas de dichas actividades. Por tanto, las *híper-aristas* del grafo de dependencias poseen los siguientes tipos:

- $\mathbf{E_{pd}}$ = ({**p**}, **O**). *F-arista* desde el nodo *p* (tipo actividad) a un conjunto de nodos *O* que contiene los datos de salida de *p*.

```
1.  INPUTS: queryGraph
2.  OUTPUT: ∅
3.  BEGIN
4.  Let StackNextNode ← ∅; QueueConnector ← ∅
5.  for each startNode ∈ StartNodes in queryGraph do
6.     state[startNode]←visited
7.     StackNextNode add startNode
8.     while StackNextNode ≠ ∅ do
9.        node ← StackNextNode remove firstNode
10.       AddQueryNode(node) /*initial node is added to the hyper-
      graph*/
11.       NeighborsList ← getNeighbors(node)
12.       for each neighbor in NeighborsList do
13.          if state[neighbor] ≠ visited then
14.             state[neighbor]←visited
15.             parent[neighbor]←node
16.             if type[neighbor] = AND-Join or
                   type[neighbor] = XOR-Join then
17.                QueueConnector add neighbor
18.             else
19.                StackNextNode add neighbor
20.             end if
21.          end if
22.       end for
23.       if StackNextNode = ∅ and QueueConnector ≠ ∅
          then
24.          StackNextNode add QueueConnector remove
             lastNode
25.       end if
26.    end while
27. end for
28. END
```

Algoritmo 2. Función DFS Modificado

- $\mathbf{E_{dp}}$ = (**I**, {**p**}). *B-arista* desde el conjunto de datos *I* al nodo *p* (tipo actividad), *I* representa el conjunto de datos de entrada de *p*.
- $\mathbf{E_{dd}}$ = ({**d**}, **D**). *F-arista* desde el nodo *d* (tipo dato) al conjunto de nodos *D* (tipo datos), este tipo de *híper-arista* se utiliza para representar la asignación del valor de un dato a otro(s).

La construcción del *Híper-grafo* de dependencias es guiada por el grafo de consulta, una representación formal del documento BPEL requerido por el usuario. Este grafo está compuesto por una lista de nodos que representan las actividades básicas requeridas para ejecutar la conversación solicitada por el usuario; para su construcción se realiza un recorrido ordenado de los nodos de consulta y se van agregando los nodos que representan los datos de entrada y/o salida de los nodos de consulta, así como los nodos recuperados. A medida que se agregan los diferentes nodos se van creando las relaciones de dependencia a través de las *híper-aristas* que conectan los vértices. Los procedimientos para la construcción del *Híper-grafo* se describen a continuación.

- *Construcción del Híper-grafo:*

Este procedimiento describe el proceso de *Recorrer el Grafo de Consulta*. Así, la construcción del *híper-grafo* sigue el orden establecido en el grafo de consulta; se hace de esta forma para establecer las relaciones de dependencia de datos y ejecución

```
1.  INPUTS: Node n; /*where n is a request node and a node n_p
    has attributes such that Op(n_p), PT(n_p), PL(n_p), AT(n_p),
    Input(n_p), Output(n_p).
2.  OUTPUT: ∅
3.  BEGIN
4.  IndexQueryNode(n) ←
    getNextPrimeNumberforQueryNodes(n)
5.  TargetSelectedNode(n) ←1
6.  If PL(n) ∈ QueryServices then
7.     QueryServices[PL(n)] ←QueryServices[PL(n)] *
       IndexQueryNode(n)
8.  else
9.     QueryServices[PL(n)]←IndexQueryNode(n)
10. end if
11. for each matchNode(n) in RS do/* Where RS is ranked list of
    service nodes (see Alg. 1)*/
12.    Let n_T ← getTargetNode(matchNode(n))
13.    IndexTarget(n_T) ←
       getNextPrimeNumberforTargetNodes(n_T)
14.    TargetSelectedNode(n) ← TargetSelectedNode(n) *
       IndexTarget(n_T)
15.    Let OutputDistance ← OutputData(n, n_T)
16.    Let inputDistance ← InputData(n, n_T)
17.    Let nodeDistance ←
       getDistance(matchNode(n))+inputDistance +
       outputDistance
18.    Hypergraph ← Hypergraph ∪ (n_T, nodeDistance)
19. end for
20. END
```

Algoritmo 3. Función Adicionar Actividad de Consulta

entre las actividades básicas. El algoritmo de búsqueda en profundidad (DFS Depth First Search) es utilizado para recorrer el grafo de manera ordenada. El Algoritmo 2 describe la función DFS modificada para recorrer el grafo de consulta. El recorrido inicia por los nodos tipo *Start* contenidos en el grafo de consulta (línea 6), agregando el nodo inicial al *híper-grafo* (línea 11), y colocando los nodos vecinos sujetos a procesamiento en una pila (línea 20). Cada nodo conserva el estado, esto quiere decir que a cada nodo procesado se le asigna el estado visitado (línea 15), con el fin de asegurar que el nodo sea procesado una sola vez. Este procedimiento es repetido por cada nodo que haya sido agregado a la pila (línea 9). Cada vez que se procesa un nodo se retira de la pila (línea 10). De manera alterna, una cola es construida (*QueueConnector*) para almacenar temporalmente los nodos tipo conector *AND-Join* y *XOR-Join* (línea 18), y así controlar la profundidad del recorrido. Encolar estos conectores en un arreglo diferente permite recorrer todas las ramas de una estructura compleja, antes de aumentar la profundidad más allá de los nodos donde convergen dichas ramas. Con esto se logra que al visitar un nodo determinado se hayan considerado todas aquellas actividades predecesoras que pertenecen a ramas concurrentes o excluyentes. El procedimiento para adicionar un nodo de consulta al *híper-grafo* de composición es detallado a continuación.

- *Adicionar nodos de consulta al Híper-grafo:*

Este procedimiento consiste en agregar los nodos tipo dato, que representan las entradas y/o salidas de la actividad básica de consulta, y sumar los nodos recuperados al *híper-grafo* creando las relaciones de dependencia entre los datos y las actividades predecesoras. El Algoritmo 3, especifica cada uno de los pasos necesarios para adicionar un nodo de consulta al *híper-grafo*.

Este algoritmo inicia asignando un código al nodo de consulta (línea 4), dicho código corresponde a un número primo único que identifica al nodo (el código es incrementado a medida que se

agregan nuevos nodos). Posteriormente, las parejas entregadas por el módulo de descubrimiento (línea 11) son recorridas con el fin de agregar los nodos recuperados al *híper-grafo*. Cada nodo recuperado es identificado por un número primo único (línea 13) lo cual permite adicionarlo al *híper-grafo* (línea 18). Mientras que los nodos recuperados son procesados, la comparación de las interfaces de entrada y salida de éstos respecto el nodo de consulta es llevada a cabo (línea 15 y 16). De esta manera son creadas las aristas de dependencias entre los datos y las actividades recuperadas. Si una actividad predecesora produce todos los datos que requiere una actividad subsecuente, se establece que son compatibles y que pueden participar en la composición.

5.2 Síntesis de Composición

La síntesis de composición selecciona el conjunto de servicios que serán empleados para realizar la tarea del usuario. Para ello, se debe seleccionar un conjunto de servicios que ofrezcan cada una de las operaciones requeridas en el proceso de negocio (tarea de usuario representada mediante un proceso descrito con BPEL) y cuyas interfaces sean compatibles. El Algoritmo 4, presenta el procedimiento para construir el árbol de servicios. Este algoritmo recibe como entrada los servicios de consulta, junto con el conjunto de servicios recuperados para cada uno de ellos (*ServicesMatches*). La construcción del árbol tiene como objetivo seleccionar un servicio recuperado n_{Tk} que pertenece a $N_T = \{n_{T1},\ldots, n_{Tp}\}$ por cada servicio de consulta n_k (ver Algoritmo 4, línea 1), para esto se van adicionando los servicios recuperados de una manera ordenada de acuerdo a su similitud con el servicio de consulta. La función *getServiceMatchForQueryService*() (línea 10) escoge de manera ordenada el servicio recuperado que debe ser adicionado al árbol. Esta selección toma en cuenta la compatibilidad del servicio recuperado con los servicios adicionados previamente. El algoritmo es ejecutado hasta que un servicio recuperado es agregado por cada servicio de consulta, esta condición asegura que todos los elementos para componer el proceso de consulta son considerados. Una vez terminada la selección de servicios, se procede a generar un proceso de negocio abstracto a partir de ellos. Este proceso es realizado siguiendo el comportamiento especificado en la tarea requerida por el usuario.

6. EXPERIMENTACIÓN

Esta sección presenta el estudio experimental de la plataforma propuesta para la composición de servicios en ambientes de computación ubicua. El algoritmo de composición se implementó en lenguaje Java y los experimentos fueron realizados en un computador con procesador Pentium 4 2,30GHz, 1.000 MB de RAM bajo el S.O. Linux, Ubuntu. Para evaluar los algoritmos definidos y determinar la eficiencia de los mismos, se caracterizaron los tiempos de respuesta según el estudio realizado en [27], donde: r es el tiempo de respuesta en segundos; de tal forma que r es clasificado como: Optimo cuando r ≤ 0.1, Bueno cuando $0.1 \leq r \leq 1$; Aceptable cuando $1 \leq r \leq 10$; y Deficiente cuando $r \geq 10$.

La evaluación de rendimiento fue realizada sobre los principales módulos que implementan las funcionalidades de la fase de composición, tales como: la conformidad de los servicios y la síntesis de la composición. De esta manera, se definieron dos casos de prueba que determinan el tiempo que tarda el módulo de composición en entregar un resultado ante una tarea de consulta, cuando: el número de servicios de consulta varía, manteniéndose constante el número servicios recuperados, y cuando se varia el

1. **INPUTS**: *ServicesMatches*/*Let *ServicesMatches* = $\{s_1,\ldots,s_n\}$ where $s_k$is a tuple of the form $S_k(n_k, N_T)$; such thatN_T= $\{n_{T1}, \ldots, n_{Tp}\}$represents an ordered set of services retrieved for the query node n_k.
2. **OUTPUT**: Tree
3. **BEGIN**
4. *Let TreeDepth/*Number of Query Services*/*
5. *Let Depth← 0*
6. *Let Tree ←∅ /*Composition Tree*/*
7. *Let TreeIndexXOR← 1 /*this index is the product of the indexes of those services that are not compatible with the services added in the composition.*/*
8. *Let TreeIndexAND← 1/* this index is the product of the indexes assigned to the services used in the composition */.*
9. **While** *Depth <TreeDepth* **do**
10. *Let serviceRetrieved ← getServiceMatchForQueryService*(*Depth*)
11. **If** *serviceRetrieved* ≠ ∅ **then**
    *TreeaddMatchServiceIn (Position←Depth) /*if this function finds a service that is compatible it is added to the tree, including its composition indices XOR and AND in order to establish what services can be added later*/*
12. *TreeIndexXOR←TreeIndexXOR * XORServices [serviceRetrieved]*
13. *TreeIndexAND←TreeIndexAND * TargetService[serviceRetrieved]*
14. *Depth++*
15. **else**
16. *Depth←Depth -1/* this is when a retrieved service is not found, which is compatible with previously selected.,in this case it returns a level in the tree to select a new service*/*
17. **If** *Depth*>= 0 **then /*If the depth of the tree is greater than zero, the node is removed and removed its composite index XOR and AND*/*
18. *RemovedService← Tree get node at Depth*
19. *TreeIndexXOR←TreeIndexXOR / XORServices [RemovedService]; TreeIndexAND←TreeIndexAND / TargetService[RemovedService]*
20. **else**
21. **return** ∅ */*When the depth is less than zero is not possible to perform the services composition */*
22. **end if**
23. **end while**
24. **return** *Tree /*this is the successful case,the tree represents a composition*/*
25. **END**

Algoritmo 4. Función Construir Árbol de Servicios

número de servicios recuperados y la cantidad de servicios que conforman el proceso de consulta se mantiene constante.

En este orden de ideas, se conformó un banco de pruebas (repositorio de procesos de negocio [22]) compuesto por documentos BPEL, distribuidos en cinco dominios relacionados con procesos de información turística: CarRental, Guesthouse, Holidays, Hotel y Travel. Estos procesos BPEL suman un total de 148 actividades básicas, las cuales representan los servicios presentes en la red ubicua y pueden ser consideradas como operaciones publicadas/disponibles para ser empleadas en las fases de descubrimiento y composición. Por tanto, para el primer caso de evaluación, se contó con 18 archivos BPEL de consulta, en los cuales se incrementó gradualmente el número de

actividades básicas contenidas en dichos procesos BPEL. Este incremento significó un aumento en el número de datos que se emplearon, tal como se aprecia en la Tabla 1, la cual expone el cálculo del número total de nodos procesados por el algoritmo de composición, al momento de crear el híper-grafo de acuerdo a los parámetros de entrada de los documentos BPEL de consulta. Este número total corresponde a la suma de la cantidad de actividades básicas y el número de datos que se declaran en el documento BPEL.

Tabla 1. Conformación de los Documentos BPEL de Consulta

BPEL	Actividades Básicas	Datos	Total Nodos	BPEL	Actividades Básicas	Datos	Total Nodos
1	3	8	**11**	10	12	47	**59**
2	4	10	**14**	11	13	50	**63**
3	5	14	**19**	12	14	30	**44**
4	6	30	**36**	13	15	34	**49**
5	7	27	**34**	14	16	53	**69**
6	8	36	**44**	15	17	56	**73**
7	9	36	**45**	16	18	56	**74**
8	10	20	**30**	17	19	59	**78**
9	11	44	**55**	18	20	62	**82**

Para el primer caso de prueba, se ingresaron los documentos de consulta a la aplicación de descubrimiento y composición, y se midió el tiempo que tarda en realizar las tareas de composición. Así, por cada documento de consulta se tomaron 5 muestras de tiempos con el fin de establecer la media y obtener una aproximación para el tiempo de ejecución. Dichas pruebas fueron realizadas en diferentes escenarios, de acuerdo al número de actividades que se deseaban recuperar del repositorio de procesos, variando el número *K* de actividades recuperadas. En la Figura 2 se muestran las gráficas de comportamiento obtenidas de la aplicación de composición para tres casos, cuando se recuperan 4, 9 y 14 actividades del repositorio de procesos. Las gráficas presentadas en la Figura 2, evidencian que el tiempo de respuesta de los algoritmos de composición varía linealmente a medida que se incrementan la cantidad de actividades básicas y los datos que pertenecen al proceso de consulta. Para dichos escenarios de evaluación (4 a 14 actividades recuperadas), la relación del tiempo de respuesta contra los nodos de entrada se puede generalizar como una recta: $\boldsymbol{t} = \boldsymbol{m}*\boldsymbol{q} + \boldsymbol{C}$; donde $\boldsymbol{t}$ es el tiempo, $\boldsymbol{m}$ la pendiente de la recta, $\boldsymbol{q}$ los nodos de consulta y $\boldsymbol{C}$ una constante. La pendiente (m) de la recta varía de acuerdo a la cantidad de actividades recuperadas, entre mayor sea el número de actividades recuperadas, mayor es la pendiente. Según la caracterización realizada en [27], se determinó los siguientes tipos de comportamiento: **Óptimo** (tiempo de respuesta menor o igual a 0,1s.) - cuando el número de nodos de consulta es inferior a 20. Esto se mantiene para todos los escenarios donde se varió el número de actividades recuperadas. **Bueno** (tiempo de respuesta entre 0,1s. y 1s.) - en esta categoría se encuentra el comportamiento del algoritmo para documentos de consulta que posean una cantidad de nodos de entrada (actividades básicas y datos) entre 20 y 82. Así, los escenarios de evaluación, con distinto número de actividades recuperadas, alcanzaron tiempos de respuesta inferiores a 1 segundo en la ejecución de los algoritmos de composición.

Para el segundo caso de estudio, se determina el comportamiento de los algoritmos de composición, variando el número de actividades básicas que conforman el documento BPEL de consulta. Estas pruebas de rendimiento fueron realizadas con 18

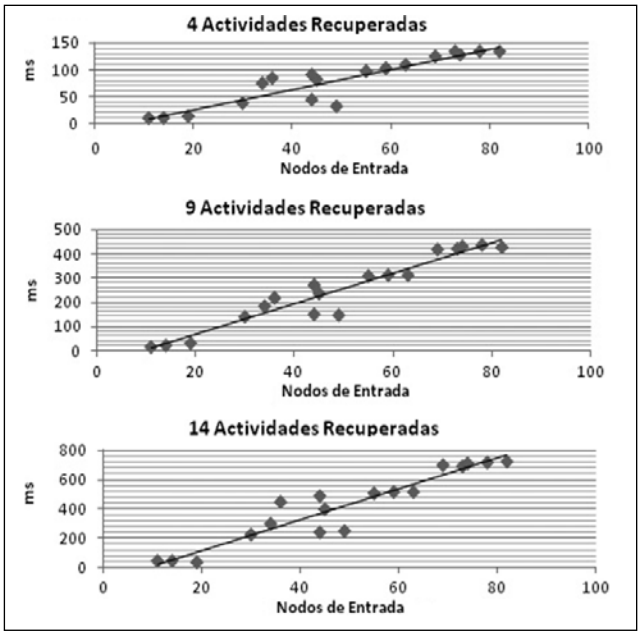

Figura 1. Gráficas de Rendimiento del Algoritmo de Composición en Función de los Nodos de Entrada

archivos (ver Tabla 1). Lo anterior, permite evaluar los tiempos de respuesta de los algoritmos a medida que se aumenta el número de servicios recuperados en un rango de 3 a 14 actividades, manteniendo constante las actividades y datos que los componen. Así, para cada incremento en el número de servicios recuperados, se toma varias muestras de tiempo, las cuales son promediadas para obtener un tiempo estimado de ejecución. Lo anterior es expuesto en la Figura 3, donde se muestran las gráficas obtenidas del comportamiento de la aplicación de composición para tres casos, procesos BPEL de consulta de 11, 59 y 82 nodos de entrada (entre actividades básicas y datos). Por tanto, las gráficas presentadas en la Figura 3, exponen el tiempo de respuesta de los algoritmos de composición, el cual varía linealmente a medida que se incrementan la cantidad de actividades básicas recuperadas. Para los escenarios evaluados (11 a 82 nodos de entrada), la relación del tiempo de respuesta contra las actividades recuperadas se puede generalizar igualmente como una recta: $\boldsymbol{t} = \boldsymbol{p}*\boldsymbol{k} + \boldsymbol{C}$; donde $\boldsymbol{t}$ representa el tiempo de respuesta, $\boldsymbol{p}$ la pendiente de la recta, $\boldsymbol{k}$ los nodos recuperados y $\boldsymbol{C}$ una constante. De esta forma, la pendiente (p) de la recta varía linealmente de acuerdo a la cantidad de actividades y datos de entrada.

7. CONCLUSIONES

El enfoque presentado usa técnicas de emparejamiento y composición que trabajan sobre modelos de comportamiento. Se realiza una evaluación de la distancia semántica entre las actividades requeridas por el usuario y aquellas contenidas en el repositorio de la plataforma, soportando la entrega de aproximaciones cuando no se obtiene una coincidencia exacta de los parámetros de búsqueda. El problema del emparejamiento de actividades y composición de servicios es abordado por medio de una representación formal de grafos, que permite extraer las actividades básicas requeridas y obtener un formalismo para el proceso de composición de las actividades recuperadas.

Considerando el comportamiento de los algoritmos de composición frente a variaciones en la cantidad de actividades de

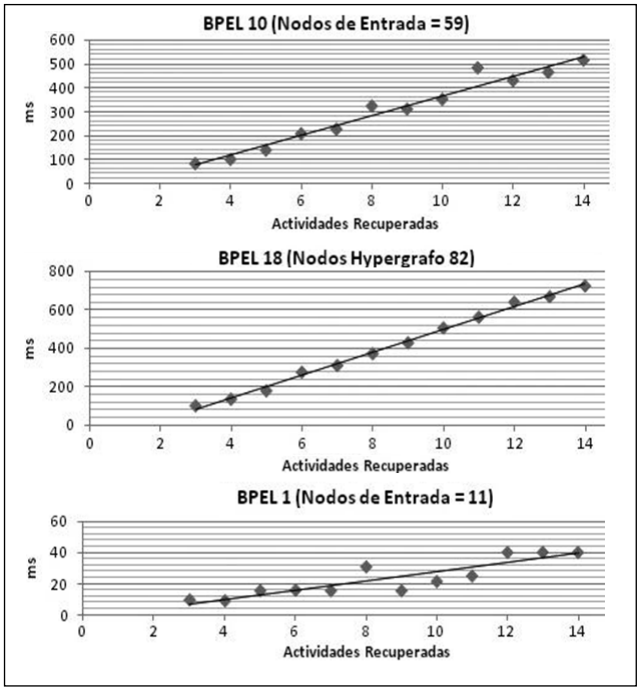

Figura 2. Gráficas de Rendimiento del Algoritmo de Composición en Función de las Actividades Recuperadas

consulta y el número de actividades recuperadas que entrega el módulo de descubrimiento, se puede concluir que la complejidad de la solución de composición está dada por: t = q*k; donde q es el número de actividades y datos de consulta, y k el número de actividades recuperadas. Así, la complejidad computacional de los módulos de descubrimiento y composición está determinada por el producto del número de actividades de consulta y la cantidad de actividades recuperadas, expresado como O(n), comportamiento esperado, y adecuado para entornos de computación ubicua, ya que los algoritmos realizan comparaciones uno a uno de las actividades que pertenecen a los dos conjuntos. El comportamiento lineal de la presente solución evidencia una diferencia significativa con respecto a los trabajos citados en la sección 2 [3, 7, 9, 13], los cuales poseen una complejidad exponencial. El número de comparaciones es reducido específicamente en la fase de descubrimiento, por medio del filtrado de candidatos, basado en el tipo de actividad, dominio de los procesos de negocio y el contexto de entrega del usuario, y por la reducción del espacio de búsqueda, gracias a los perfiles de usuario. Como trabajo futuro se plantea abordar la fase de orquestación de la composición, la cual permitirá realizar un control dinámico de las síntesis de composición generadas durante la fase de ejecución. Con esto es posible obtener capacidades de reconfiguración de los planes de composición sobre la marcha, una vez iniciada la ejecución de las tareas de usuario.

8. REFERENCIAS

[1] Suarez, L. J., Rojas, L. A., Corrales, J. C. and Steller, L. A. 2011. *Service Discovery in Ubiquitous Computing Environments*. In Proceedings of the The Sixth ICIW. St. Maarten.

[2] El-Sayed and Abdur-Rahman. 2006. *Semantic-based context-aware service discovery in pervasive-computing environments*. In of the Proceedings of 1st IEEE SIPE, Lyon, France.

[3] Mokhtar, S. B. 2007. *Semantic Middleware for Service-Oriented Pervasive Computing*. Ph.D Thesis, Devant L'universit´ e de Paris 6, Paris, Francia.

[4] M. Sellami, S. T., B. 2009. *Defude Service Discovery in Ubiquitous Environments: Approaches and Requirement for Context-Awareness*. BPM Workshops, Venice, Italy.

[5] Hermida, V., O. Caicedo, J.C. Corrales, D. Grigori, and M. Bouzeghoub. 2009. *Service Composition Platform for Ubiquitous Environments Based on Service and Context Matchmaking*. 4CCC, Bucaramanga.

[6] Daniela Grigori, J. C. C., Mokrane Bouzeghoub, Ahmed. 2010. *Gater Ranking BPEL Processes for Service Discovery*.

[7] Zhenghui, W., Xu Tianyin, Q., Zhuzhong and Sanglu, L. A. 2009. *Parameter-Based Scheme for Service Composition in Pervasive Computing Environment*. In Proc. of the CISIS, Fukuoka, Japan.

[8] Corrales, J. C. 2008. *Behavioral matchmaking for service retrieval*. Tesis presentada a la University of Versailles Saint-Quentin-en-Yvelines para optar al grado de Doctor of Philosophy in Sciences, Versailles, France.

[9] Berardi, D., Calvanese, D., Giacomo, G. D., Hull, R. and Mecella, M. 2005. *Automatic Composition of Web Services in Colombo*. In Proceedings of the SEBD, Brixen-Bressanone, Italy.

[10] Brogi, A., Corfini, S. and Popescu, R. 2008. *Semantics-based composition-oriented discovery of Web services*. In Proceedings of the ACM Transactions on Internet Technology (TOIT).

[11] Andrews Tony, C. F., Dholakia Hitesh, Goland Yaron, Klein Johannes, Leymann Frank, Liu Kevin, Roller S.A. Dieter, Smith Doug, Thatte Satish, Trickovic Ivana, Weerawarana Sanjiva. 2003. *Business Process Execution Language Version 1.1. the BPEL4WS Specification*.

[12] Küster, U., Stern, M. and König-Ries, B. 2005. *A classification of issures and approaches in service composition*. In Proceedings of the In Proceedings of the WESC05, Amsterdam, Netherlands.

[13] Hashemian, S. V. and Mavaddat, F. 2005. *A Graph-Based Approach to Web Services Composition*. (SAINT'05).

[14] Ponnekanti, S. R. and Fox, A. 2002. *SWORD: A developer toolkit for web service composition*. In Proceedings of the The 11th Int. WWW Conf. (WWW2002), Honolulu, USA.

[15] Martínez, E. and Lespérance, Y. 2004. *Web service composition as a planning task: Experiments using knowledge-based planning*. In Proceedings of the the 14th Int. Conf. on Automated Planning and Scheduling (ICAPS 2004), Whistler, BC, Canada.

[16] Masuoka, R., Parsia, B. and Labrou, Y. 2003. *Task computing the semantic web meets pervasive computing*. In Proc. of the ISWC2003.

[17] Ramasamy, V. 2006. *Syntactical and semantical web services discovery and composition*. CEC/EEE'06.

[18] Zhang, R., Arpinar, I. B. and Aleman-Meza, B. 2003. *Automatic composition of semantic web services*. ICWS, Las Vegas, NV, USA.

[19] Bernstein, A. and Klein, M. 2002. *Towards high-precision service retrieval*. In Proceedings of the ISWC, LNCS, Sardinia. Italia.

[20] Shiaa, M. M., Fladmark, J. O. and Thiell, B. 2008. *An Incremental Graph-based Approach to Automatic Service Composition*. In Proceedings of the SCC'08, Washington, DC, USA.

[21] Hackmann, G., Gill, C. and Roman, G. C. 2007. *Extending BPEL for interoperable pervasive computing*. In Proceedings of the ICPS, Istanbul.

[22] Vanhatalo, J., Koehler, J. and Leymann, F. 2006. *Repository for business processes and arbitrary associated metadata*. In Proceedings of the BPM Demo Session at the Fourth International Conference on Business Process Management, Viena. Austria.

[23] Wireless Application Protocol Forum, WAG UAProf http://www.wapforum.org/, 2001.

[24] Passani, L. Wurlf. http://wurfl.sourceforge.net/, 2007.

[25] Mendling, J. and Ziemann, J. 2005. *Transformation of bpel processes to epcs*. In Proc. of the EPK2005, Hamburg. Germany.

[26] Gallo, G., Longo, G., Nguyen, S. and Pallottino, S. 1993. *Directed hypergraphs and applications*. In Proceedings of the Discr. Appl. Math.

[27] Joines, S., Willenborg, R. and Hygh, K. 2002. *Performance Analysis for Java Websites*. Addison-Wesley, ISBN-13: 978-0201844542.