Service Discovery in Ubiquitous Computing Environments

Luis Javier Suarez
GIT
University of Cauca
Popayán, Colombia
ljsuarez@unicauca.edu.co

Luis Antonio Rojas
GIT
University of Cauca
Popayán, Colombia
luisrojas@unicauca.edu.co

Juan Carlos Corrales
GIT
University of Cauca
Popayán, Colombia
jcorral@unicauca.edu.co

Luke Albert Steller
Faculty of I.T.
Monash, University
Victoria, Australia
Luke.Steller@gmail.com

***Abstract*—Today, there is an increasing abundance of information and services available to mobile users. Many ubiquitous services retrieval architectures are based on keyword or interface matching which does not provide very accurate match results. More recently, semantic languages have been used to improve accuracy. However, this often requires the use of reasoning software which is very resource intensive. Therefore, in this paper we propose a semantic approach to service retrieval in ubiquitous computing environments, which improves accuracy over keyword / interface matching approaches but avoids the use of a semantic reasoned in order to provide improved efficiency over inference based proposals. In addition, our proposal incorporates a user profile to limit the search space and takes account of the capabilities of the requesting mobile device. Our approach also transforms BPEL service descriptions into a graph to perform atomic-level graph matching. Thus, we calculate semantic similarity between two graph nodes to provide a service ranking, so that it is possible obtain an approximate match if there is no service that exactly matches the user requirements. We have implemented our approach and provide a performance evaluation on a mobile device which clearly demonstrates that our approach is more efficient than reasoning and produces accurate match results.**

Keywords-matching; context-aware discovery; ubiquitous environments; personalization

I. INTRODUCTION

The number of mobile subscribers is reaching the 3 billion mark, worldwide [1]. The vision of ubiquitous computing is the amicable integration of small devices, computing and communication capabilities with humans [2] to assist them in performing their tasks, anytime and anywhere. The goal is for this integration to be as seamless as possible, ideally unconscious to the human user. Service oriented architectures [3], are useful to support transparent integration of software applications in ubiquitous environments [4]. Service discovery is used to match the requirements of a mobile user with the capabilities of existing services available. Since ubiquitous mobile environments are extremely dynamic, this matching process must be both accurate / relevant [5] and fast /efficient [6].

Service discovery in ubiquitous environments presents both new opportunities and new challenges [7, 8]. On one hand there is an abundance of contextual information about the mobile which can enrich the service discovery process. On the other hand mobile devices used in ubiquitous environments are typically resource constrained and cannot interact with all services.

In this paper we propose a service discovery architecture for ubiquitous environments which considers the preferences of mobile users, the resource specifications of the user's device and the delivery context to provide the flexibility to reconfigure services according to environmental changes.

Typically the Business Process Execution Language (BPEL) [9] is used as an orchestration language for services. It is used to form executable business processes which involve message exchange. The number of business processes described using BPEL on the web and at an enterprise level is increasing. Additionally, BPEL is useful for forming a composition of multiple services to meet the user's requirements when a single service alone cannot perform the required task [10]. Therefore, in our approach we propose an algorithm which matches services based on BPEL descriptions.

It is well known that semantic matching is more accurate than earlier approaches such as keyword / interface based matching [11, 12]. Therefore, in order to meet the need for accuracy, our matching algorithm evaluates semantic distance between existing services. Many semantic matching approaches utilize reasoners, however, the use of reasoners has been shown to be extremely resource intensive [3, 13-15]. Therefore, in order to support efficiency we avoid the use of reasoners. Rather, we reduce the matching process to a problem of graph matching by adapting existing algorithms [16, 17]. As such our matching algorithm translates BPEL processes into graph representations then matches these graphs using semantic distance calculations [17].

We have implemented our proposed approach and provide an evaluation on a resource constrained device which shows that our approach supports both efficient matching on a resource constrained device and effectively provides accurate results.

The remainder of this paper is structured as follows: A discussion of the current research in the field is given in Section II. We present the high-level description of our architecture and matching process in Section III. Then in Section IV we discuss our approach to transform BPEL into graphs. The overall ranking process is discussed in Section V, followed by details about how two graph nodes are compared in Section VI. In Section VII we discuss the way in which our architecture filters services based on whether they are capable of running on the user's device. We provide

 ISBN: 978-1-61208-124-3

details about our implementation and evaluation in Section VIII. Finally in Section IX we conclude the paper.

II. RELATED WORK

Service discovery is defined as the ability to find and use a service based on a published description of its functionality and operational parameters[18]. Service discovery can be addressed under two main approaches: syntactic and semantic discovery.

Syntactic discovery is based on interface matching techniques (e.g., UDDI, ebXML, WSDL, IDL, RMI interfaces, etc.) or keywords to search for services, requiring exact matches at the syntactic level between service descriptions and parameters employees [7, 19, 20], which can result in that equivalent services at the logical level to be discarded (e.g., two services described as *printer* and *printing* may be differ syntactically but logically they are equivalent).

Thus, while the syntax is focused on defining the services from the input and output messages, types and parts of the message, semantics aims to provide information about the service functionality[19, 21].Thus, semantics improves matching accuracy. The semantic representation of service descriptions content enable machines to understand and process their content, supporting the discovery and service dynamic integration[7]. However, semantic descriptions require reasoning applications which are resource intensive applications which will significantly increase processing time[22].

Therefore, we propose a service discovery approach for ubiquitous environments based on semantic matching without a reasoner. Our approach provides a ranked list of services which completely or partially match a user request. In addition, service retrieval process considers the preferences of mobile users, the resource specifications of the user's device and the delivery context to provide the flexibility to reconfigure services according to environmental changes.

III. ARCHITECTURE AND MATCHING PROCESS

In this section we describe our proposed architecture to perform semantic service discovery in ubiquitous environments by considering the user request, user profile and device context. In our approach, which is named U-ServiceMatch, services and user requests are described using BPEL. Figure 1 depicts our architecture which is composed of the following modules:

- *Advertiser:* Service providers advertise their services as BPEL documents, to the *Advertiser Module*, which stores this service description into the *Service Repository*.
- *Requester:* A service requesters is a mobile user which submits a BPEL request for a service.
- *BPEL Parser:* This module transforms a BPEL service description or user request into a graph, and vice versa.
- *Device Repository:* This repository stores the resource capabilities of the requesting user's device, including processing power, screen size, input interface, etc.
- *User Repository:* This module stores details related to the mobile user / requester including personal information about the user and previously requested / invoked services.
- *Service Discovery:* This module performs the matching of a user request to service descriptions. It contains several sub-modules including the *Service Matcher* which performs the graph matching, the *Context Matcher* which determines whether services can be displayed on the user's device and *User Matcher* which matches user profiles.

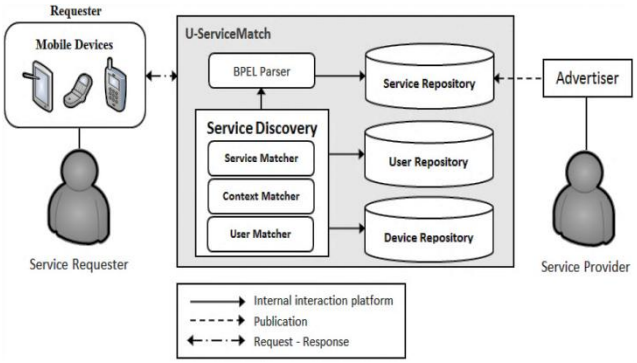

Figure 1. Architecture U-ServiceMatch

The overall module interaction is presented as an activity diagram in Figure 2. This can be described as follows.

A user submits a BPEL service description which is transformed into a graph by the *BPEL Parser*. The *Service Discovery* module then manages the matching process as follows. The user request graph is first matched (by the *Service Matcher*) with services that have been consumed in the past by the current user or other users with a similar user profile as the current user. Similar users are found by the *User Matcher* module. This step is designed to limit the search space. If a sufficiently matching service was not found, then the user request is matched by the *Service Matcher* against all other services in the *Service Repository*. Each service in the ranked list of services is checked to ensure it can be invoked / consumed by the requesting device by the Context Matcher. A final ranked service list is provided to the requester.

In the remainder of this paper we will discuss the following. In Section IV we will discuss the BPEL to graph transformation which is handled by the *BPEL Parser* module. In Section V we will present the overall ranking process and user profile matching handled by the *Service Discovery* module which will interact with the *User Matcher* sub-module, and the *User and Device Repositories*. In Section VI we will discuss how two graph nodes are compared by the *Service Matcher* module. In Section VII we will talk over the way in which our *Context Matcher* filters services based on whether they are capable of running on the user's device.

 ISBN: 978-1-61208-124-3

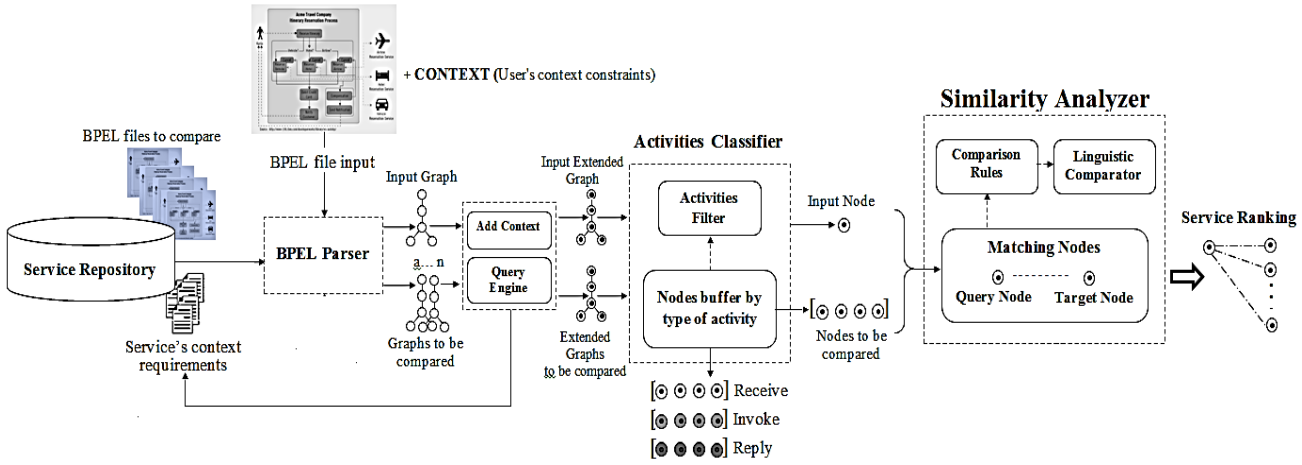

Figure 2. Matching of BPEL Basic Activities

IV. BPEL TO GRAPH TRANSFORMATION

In this work the available services in the ubiquitous network are represented by basic activities contained in a business process, denoted by BPEL. Thus, in this section, we will discuss how BPEL processes are transformed to graphs. Similarly, the nodes of the transformed graph represent the activities of the BPEL process.

*Transforming BPEL to Graph:*Graphs are a general and powerful data structure for representing objects and concepts. Thus, in this section will present the equivalence between a BPEL description and a formal representation of Graphs.

A graph G, in its basic form, is a pair $G = (N, E)$ where N is a non-empty finite set of elements called *nodes* (also called vertices or points) such that $N = \{n_1, \ldots, n_m\}$.E is a multi-set of pairs (n_i, n_k) is not ordered distinct elements of N called *edges,* such that and $E \subset N \times N$. N and E are distinct, such that $N \cap E = \varnothing$. When all the edges have directions, and therefore (n_i, n_k) and (n_k, n_i) can be distinguished, the graph is directed. Thus, a *directed graph* or *digraph*$G = (N, E)$ consists of a set N of nodes and a set E of edges, which are ordered pairs of elements of N.

The *BPEL Parser*module transforms a BPEL behavior model into a process graph. A process graphhas at least one start node and can have multiple end nodes.The graph can have two kindof nodes:(1) regular nodes representing BPEL activities; and (2) BPEL connectors representing splitand join rules of type XOR or AND. Nodes are connected via edges which may have anoptional guard. Guards are conditions that can evaluate to *true* or *false*.

We used the flettening strategy presented in [23] to transform a BPEL document to a process graph. The general idea is to map structured activities to respective process graph fragments, Figure 3. The algorithm traverses the nested structure of BPEL control flow in a top-down manner and recursively applies a transformation procedure to each type of structured activity.

A BPEL *basic* activity is transformed into a graph node *n*. The BPEL *sequence* is transformed by connecting all nested activities with graph edges; each sub-activity is then transformed recursively. For the BPEL *while* activity, a loop is created between an *XOR join* and a BPEL*XOR split*, the condition is added to the edge. The graph representation of BPEL *switch* consists of a block of alternative branches between a BPEL*XOR split* and a BPEL*XOR join*. The branching conditions are each associated with an edge. The BPEL *flow* is transformed to a block of parallel branches starting with a BPEL *AND split* and synchronized with a BPEL *AND join*.

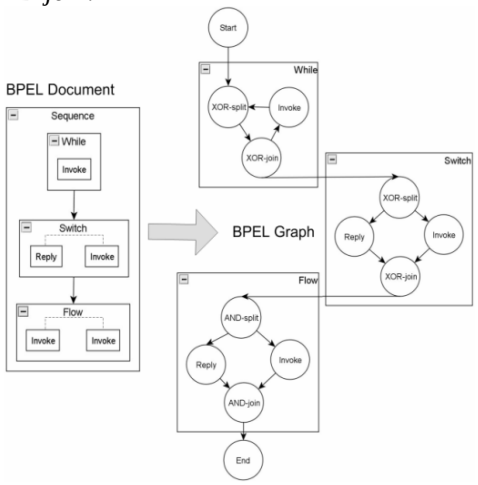

Figure 3.Correspondence between BPEL elements and Graph elements

The graph nodes *n* that represent BPEL activities have attributes which reflect the respective activity. These are defined as ActivityType *AT*(*n*), Operation *Op*(*n*), PortType *PT*(*n*) and PartnerLink *PL*(*n*). *AT*(*n*) may contain one of the following values $Invoke^{syn}$, $Invoke^{asyn}$, *Receive* or *Reply*. The graph nodes *n* that represent BPEL connectors have two attributes defined as: *ConnectorType*(*n*) and *ActivityType*(*n*).

ConnectorType(*n*) may contain one of the following values: *AND-split*, *AND-join*, *XOR-split* or *XOR-join*. *ActivityType*(*n*) is the BPEL structured activity from which the node was derived during transformation. Figure 3 shows the correspondence between BPEL constructs and graph elements.

V. USER PROFILE MATCHING AND SERVICE RANKING

To produce a ranked set of services the mobile user's service request node n_i must be matched against each service node n_j contained within a set *S* of potential services. There may be many potential services in the *Service Repository*. Therefore, we first check if any user has performed the same request previously, and if so obtain a ranked service list from the cache. If the request is not in the cache, the matching algorithm matches the user request against those services which have been invoked previously by the same user or a different user which has a similar user profile as the current user. If a valid service has still not been found, then the remaining services in the *Service Repository* are compared against the request.

This process is the focus of this section. First we will describe the structure of our user profiles then secondly we will describe the matching algorithm which provides a ranked list of services.

A. User Profile Structure

The structure of our user profiles is based on [24]. These profiles comprise domain of interest and personal data as shown in Figure 4. In this paper, we present a proof of concept which takes a few of these characteristics into consideration. In future work, we will expand the contextual attributes which are taken into consideration to provide a broader matching of user profile similarity.

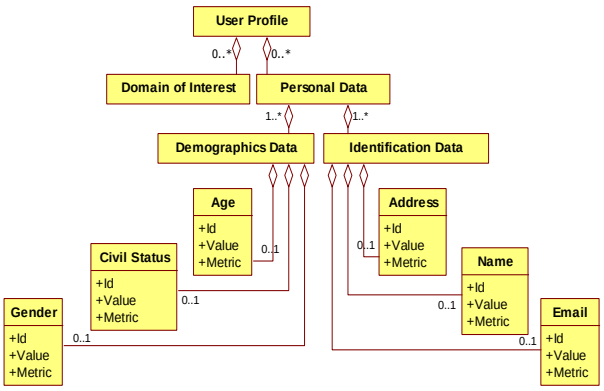

Figure 4. Meta-Model of User Profile

Several studies use different methods for collecting and handling domain of interest information, depending on the application: Web mining [25], clustering [26], Application logs [27], etc. Each of these mechanisms generates a set of parameters and their possible values for a given domain of interest. The definition of these parameters and values are not established in this work, due to the high level of analysis and decoupling to a specific field.

Personal data falls into two categories: data identification and demographics. The user profile meta-model, is stored in the *User Repository*. In our system, we compare a user profile with other profiles in order to establish a set of similar user profiles. We assume that users with similar profiles will request similar services [28]. Thus, we suggest services to a user if these have been requested or consumed by a similar user in order to reduce the search space for potential services to compare against the user request. To realize this goal, we will propose the matching process in the next subsection.

B. Rank Services

Algorithm 1 defines the algorithm for obtaining a ranked set of services which match the user request. This algorithm makes use of two functions. Let the function *GetRankedServicesFromCache*(*n*) provide a ranked list of services from the cache for any user request *n*(if one exists). If the current or another user has not submitted the request *n* previously, then the algorithm retrieves a list of services which the current user, or other users with a similar user profile, has invoked in the past. Let *ConsumedServices*(p_i) denote a function which returns these services, where p_i is the user profile for the current user / requester.

Algorithm 1. RankServices

```
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 Alg. 3, Sec. VI*/
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 Alg. 3 Sec. VI */
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. return RS
27. END
```

The algorithm will obtain a match result by comparing the user request node n_i against each of these previously invoked services and add these to a ranked list. The *CheckMatch*(n_i, n_p) is a function which returns a double indicating the semantic similarity / distance between the

request node n_i and a service node n_p, which will be defined in Algorithm 3, Section VI.

Assume *ConsumedServices* passes each (p_i, p_j) pair to *CheckProfileMatch* which is defined in Algorithm 2, where p_i is the user request and p_j is all user profiles in the *User Repository* module. Algorithm 2 compares the age, marital status, gender and all interest domain attributes, associated with the two user profiles, using the algorithm *LS*, which will defined in Algorithm 4 in Section VI.

Let *BadSuggest(RS)* denote a function which returns true if a given ranked list of services *RS*, does not contain enough services which meet a semantic similarity threshold against the user request. This condition is set by the requesting user. In the case that a service which satisfactorily matches the user request was not found (i.e. *BadSuggest* returns true), then all other services in the *Service Repository* will be compared with the service request to produce a ranked service list. Let *LookupServiceRepository* denote a function which returns the services from the *Service Repository*.

Algorithm 2.CheckProfileMatch

1. **INPUTS**UserProfile p_i, UserProfile p_j /* where a p_k has attributes: Set *InterestDomains*(p_k), int *Age*(p_k), String *Marital*(p_k), String *Gender*(p_k) */
2. **OUTPUT**: double
3. **BEGIN**
4. *Let* $m \leftarrow 0$, $g \leftarrow 0$, $maxI \leftarrow 0$
5. Let $a = 1-\{[|Age(p_i) - Age(p_j)|/[(Age(p_i)+Age(p_j))/2]\}$
6. **if** $Marital(p_i) = Marital(p_j)$**then,** $m \leftarrow 1$
7. **if** $Gender(p_i) = Gender(p_j)$**then,** $g \leftarrow 1$
8. **for** each value v_a**in** $InterestDomains(p_i)$ **do**
9. **for** each value v_b **in** $InterestDomains(p_j)$ **do**
10. **if** $LS(v_a,v_b) > maxI$ /* calculate similarity of p_i and p_j*/ **then**
11. $maxI = LS(v_a,v_b)$
12. **end if**
13. **end for**
14. **end for**
15. */Let w(y)* be a user assigned weight of importance where *y* is an attribute, $Age(p_i)$, $Marital(p_i)$, $Gender(p_i)$ or $InterestDomains(P_i)$, such that $0 \leq w(y) \leq 1$ */

$$\textbf{return} 1-\frac{w(Age(p_i))*a + w(Marital(p_i))*m + w(Gender(p_i))*g + w(IntegerDomains(p_i))*maxI}{w(Age(p_i)) + w(Marital(p_i)) + w(Gender(p_i)) + w(InterestDomains(p_i))}$$

16. **END**

In the next section we will define the *CheckMatch* function which calculates the semantic similarity between two graph nodes.

VI. ATOMIC-LEVEL GRAPH MATCHING

Matching the user request to a potential service involves the matching of two BPEL activities (as was shown in Figure 2). Let the request / query graph be denoted as G_Q and a service / target graph as G_T. Before running the matching algorithm for the nodes (n_i, n_j) where $n_i \in G_Q$ and $n_j \in G_T$, we organize / filter nodes (n_i, n_j) according to their BPEL activity type (this is completed by the *Activities Classifier* action in Figure 2). Therefore, only the nodes that belong to the same activity type in G_Q and G_T, respectively, are compared.

The organized nodes are then compared for matching (this is completed by the *Similarity Analyzer* module shown in Figure 2). A pair of nodes (n_i, n_j) are compared by considering their semantic distance which is outlined in Algorithm 3. This algorithm also makes use of Algorithm 4 which determines the linguistic similarity between two nodes and returns a value between 1 and 0, where 1 denotes a complete match.

Algorithm 3 starts by giving priority to comparison of the operation attribute. If the two operation attributes are similar it continuing with the calculation of the similarity of other parameters (i.e. port type and partner link) to estimate the semantic distance between the two activities. In the algorithm, let $w(Op(n_i))$, *or* $w(PT(n_i))$, $w(PL(n_i))$, denote user specified weights of importance associated with $Op(n_i)$, $PT(n_i)$, $PL(n_i)$ in the user request, respectively.

Algorithm 3. CheckMatch

1. **INPUTS**: **Node** n_i, **Node** n_j:/* where n_i is a request node and n_j is a service node and a node n_p has attributes such that $Op(n_p)$, $PT(n_p)$, $PL(n_p)$, $AT(n_p)$ as defined in Section IV */
2. **OUTPUT**: *double*
3. **BEGIN**
4. $OPS \leftarrow LS(Op(n_i), Op(n_j))$/* Operation Similarity (see Alg. 4)*/
5. **if** *OPS* = 0 (different Operations) **then**
6. **return** 1
7. **else**
8. *Let* $PTS \leftarrow LS(PT(n_i), PT(n_j))$/*PortType Similarity (see Alg. 4)*/
9. *Let* $PLS \leftarrow LS(PL(n_i), PL(n_j))$/* PartnerLink Similarity (see Alg. 4)*/
10. */w(z)* is a weight of importance associated with an attribute *z* in the user request, such that $z = Op(n_i)$, *or* $z=PT(n_i)$, *or* $z=PL(n_i)$, where $0 \leq w(z) \leq 1$*/

$$\textit{Let dist} \leftarrow 1-\frac{w(Op(n_i))*OPS + w(PT(n_i))*PTS + w(PL(n_i))*PLS}{w(Op(n_i)) + w(PT(n_i)) + w(PL(n_i))}$$

11. **Return** *dist*
12. **end if**
13. **END**

The *LS* function is defined in Algorithm 4 and is used to calculate the linguistic similarity of the values associated with the same attribute of two separate graph nodes n_i and n_j (e.g., the value of $Op(n_i)$ compared to the value of $Op(n_j)$).

Algorithm 4.LS /* LinguisticSimilarity */

1. **INPUTS**: String v_i, String v_j
2. **OUTPUT**: *double*
3. **BEGIN**
4. $$LS=\begin{cases} 1 & \text{if } (m1=1 \vee m2=1 \vee m3=1) \\ m2 & \text{if}(0<m2<1 \wedge m1=m3=0) \\ 0 & \text{if } (m1=m2=m3=0) \\ \frac{m1+m2+m3}{3} & \text{if}(m1, m2, m3 \in (0,1)) \end{cases}$$

where $m_1 \leftarrow NGram(v_i, v_j)$, $m_2 = CheckSynonym(v_i, v_j)$, $m_3 = CheckAbbreviation(v_i, v_j)$ /* *see* [29] */

5. **return** *LS*
6. **END**

In this algorithm, let *NGram*, *CheckAbbreviation* and *CheckSynonym* denote measures which are defined in [29].

NGram algorithm estimates the similarity according to a common number of *q-grams* (a *q*-gram in this context refers to a sequence of letters, *q* letters long, from a given word) between the tags. *CheckSynonym* algorithm use WordNet [30] linguistic dictionary to identify synonyms, It groups English words into sets of synonyms called synsets. Synsets are interlinked by means of conceptual-semantic and lexical relations. The *CheckAbbreviation* algorithm uses a dictionary of abbreviations appropriate to the application domain. If all algorithms give a value of 1, then there is an exact match between the tags. If all give a value of 0, then there is no similarity between words. If the values produced by *CheckAbbreviation* and *Ngram* are equal to 0 and *CheckSynonym* value is between 0 and 1, the total value of the similarity is equal to *CheckSynonym*. Finally, if all three algorithms yield a value between 0 and 1, the linguistic similarity is the average of the three.

VII. CONTEXT MANAGEMENT

Since mobile users carry their device with them throughout their daily travels, there is an abundance of contextual data available which can be fed into the service matching process to provide more accurate search results [22, 31]. Our architecture captures the resource capabilities of the requesting user's device and the resource requirements for each service. The user's device capabilities are stored in our *Device Repository* and the service requirements of each service are stored in the *Service Repository*. After the matching process defined in the previous sections of this paper, each service in the ranked list are checked to ensure they will function on the user's device. In the remainder of this section we will describe the structure of user context followed by the use of this information in the service ranking process.

A. User Context Structure

We capture user context characteristics such as processing power, modes of presentation, input interfaces, connectivity, etc. According to [24] context constraints, are defined as any information that could be used to characterize an entity, where an entity can be a person or object that is considered relevant to the interaction between user and an application. We propose three dimensions for defining a meta-model of user's context:

a) Spatial Dimension: contains all the parameters that are associated with geographical and spatial information of the user;

b) Temporal Dimension: contains the date and time of when a service is invoked;

c) Device DataDimension: contains information related to the user's mobile device such as installed software, operating system, processing power, available memory, etc. We capture this content using a CC/PP profile [32].

These dimensions are illustrated in Figure 5.

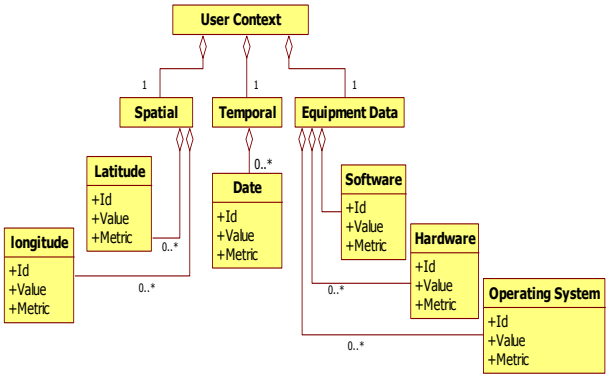

Figure 5. Meta-Model of User´s Context

The *Service Repository*, supported by the work presented in [33], stores BPEL documents and other XML files which capture the business process of services with context features. We define an XML meta-data, a model based on EMF (Eclipse Modeling Framework) for describing the restrictions specified by service providers or service developers.

The *CheckDeliveryContext* function, defined in Algorithm 5, obtains user's context and the requirements of a particular service.

Algorithm 5. CheckDeliveryContext

```
1.  INPUTS: listRankedServices
2.  OUTPUT: Set rankedFilteredServices
3.  BEGIN
    DeviceProfile deviceProfile←LookupDeviceProfile()
4.  for each nj in listRankedServices do
5.      EmfContext serviceContext←LookupFeatures.Context(nj)
6.      for each ck in serviceContext do
7.          if ck ∈ deviceProfile/* requirement supported */then
8.              rankedFilteredServices← rankedFilteredServices ∪ nj
9.              break for
10.         end if
11.     end for
12. end for
13. return rankedFilteredServices
14. END
```

Algorithm 5 takes set of ranked services obtained during the matching phase, and checks each service to see whether it meets the requirements of the service context retrieved from the *Service Repository*. Let *LookupDeviceProfile* be a function which returns the device profile for the current device (i.e., from the *Device Repository*). Let *LookupFeatures.Context(n_j)* return the context requirements for a service n_j (i.e., from the *Service Repository*). In the case that the current user's device can support the current service it's added to the set which is returned, otherwise, it is discarded.

VIII. IMPLEMENTATION AND EXPERIMENTATION

This section presents the implementation and experimental study of our proposed service matching scheme for ubiquitous computing environments. Our prototype was implemented in Java. Our experiments were

completed on the following machines / devices. The server application was running on a Pentium 4, 2.30GHz processor, 1,028 MB of RAM under the OS Linux Ubuntu. We performed tests using two real client devices / phones and two emulators. The specifications for each are provided in Table I.

TABLE I. TECHNICAL SPECIFICATION OF DEVICES USED IN EACH TEST.

Device	Processor	RAM	ROM	Screen Size	Operating Systems
Pocket PC DELL AXIM x51v	Intel PXA270, 520 MHz	64M B	256M B	480 X 640 Pixels.	Microsoft Windows Mobile 5.0
Nokia N93	Dual ARM 11 332 MHz	64M B	50 MB	128 X 160 Pixels.	Symbian 9.1
Nokia 6212 NFC			Series 40 5th Edition emulator SDK		
Nokia 6260			Series 40 6th Edition emulator SDK		

A. Evaluation Methodology

We evaluated our architecture to ensure that it is both efficient and accurate. We categorized response time efficiency as follows. Let r denote response time in seconds. Let response time be classified as: *Optimal* where $r \leq 0.1$,*Good* where $0.1 \leq r \leq 1$; *Acceptable* where $1 \leq r \leq 10$; and *Deficient* where $r \geq 10$ [34]. Accuracy was measured by comparing a set of expected values against the results obtained from our architecture, using the calculations of Precision, Recall and Overall [11, 35]. Precision p is a measure of whether the list of matching services returned by our approach contains any services which were not expected to match, such that $p = x/N$, where x denotes number of services which were both expected and proven to match and N denotes the number of services found to match. Recall r is a measure of whether all of the services which were expected to match are contained in list of matching services returned by our architecture, such that $r = x/n$, where n denotes the number of services which were expected to match. The overall o match result takes account of both precision and recall such that, by $o = r * (2 – 1 / p)$.

In our evaluation we created and compared 30 BPEL basic activities against 144 activities stored in the *Service Repository*, resulting in 1106 pairs to evaluate. The evaluations were done by 5 experts in service discovery, resulting in 5530 comparisons. These comparisons evaluate the attribute similarity between two BPEL basic activities. The human evaluator first made a comparison between the activities, and assigned an expected score to each activity according to their similarity to each user request, using our benchmarking tool [36]. Let s denote this score, such that $0 \leq s \leq 5$ where 0 implies no similarity / match and 5 implies complete similarity / match. The expert evaluator also sets the weights $w(z)$ for each compared attribute z to determine these expected results, which are also associated with the user requests being compared against the services in the actual system (see Algorithm 3, Section VI). The values obtained during our results were calculated using the micro-averaging technique [35].

B. Results

In this section we present the results from our tests.

1) Performance Evaluation (efficiency)

Figure 6 presents the execution times of our architecture for each of the different mobile client devices.

In each test, there were 17 BPEL files published in the *Service Repository* containing144 target nodes or basic target activities. In addition, 5 BPEL files were used to represent 5 separate user request queries, which were each compared with the 144 target nodes.

All tests completed on the mobile devices produced results in less than 1 second for up to 144 nodes, meaning the behavior was **good**. These tests also show that our approach is substantially more efficient than using semantic reasoners which are resource intensive. For instance, in other research we used ontologies BPMO (Business Process Modeling Ontology), eTOM (enhanced Telecom Operations Map) and SID (Shared Information/Data)[37] described in WSML (Web Service Modeling Language) [38]and performed an inference / matching task on the WSML2Reasoner reasoner[39]and found that a reasoning task required approximately 170ms for just one task [13]. Our approach performed 8 comparisons in this time on the real devices (which includes network latency) and over 32 comparisons using the emulator.

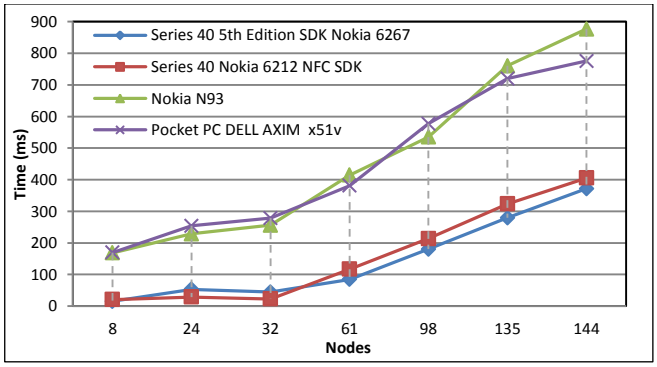

Figure 6. Recovery process performance of services on different mobile terminals.

Additionally, research shows that evaluating the control flow of BPEL documents can be exponential [17]. Our evaluation shows that our approach overcomes this problem, providing more linear results. If we extrapolate the average response time for the two real devices (i.e. Nokia and Pocket PC) presented in Figure 6 linearly[34], we can say that our architecture will have the following behavior: **Good**: when the number of graph node comparisons are less than 374.3. **Acceptable**: when the number of graph node comparisons are greater than 374.3 and less than 4145.8. **Deficient**: when the number of graph node comparisons completed are greater than 4145.8.

2) *Quality Test Results (efficacy):* in the following we present a simulation of the service matching process on a Nokia 6260 Emulator.

Figure 7. Effectiveness Test Emulator Nokia 6260: (a) service request, (b) retrievedservice (c) service selection.

In Figure 7(a) we provide an option to select one of the 5 user request queries to compare against the available services. As shown in Figure 7(b) the user receives a listing of services which are semantically similar to the user request which was selected. In Figure 8(c) the user selects the most appropriate service from the ranked list of semantically similar services.

In Figure 8, we present the precision, recall and overall match results for our tests. A precision, recall or overall match results of 1 means that the results obtained from our architecture were equivalent to the expected results. A result of 0 means that none of the expected results were obtained.

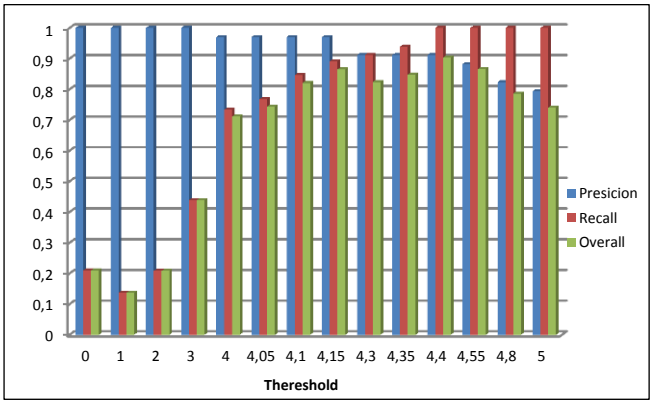

Figure 8.Quality of results produced by the U-ServciceMatch Platform

The x-axis on the graph indicates the expected similarity value *s* defined earlier in this section. Each bar shows an average of the precision/recall/overall results returned by U-ServiceMatch for all services with the same expected result *s*. We observe that services which had an expected match result of *s*=4.4 had the best precision, recall and overall match results (i.e., at least0.9 for each). We observed that while precision was high in all tests, a recall level above 0.7 was only achieved when the threshold value was *s*=4 or above. The results also show that our approach effectively supports approximate matching of a service description with a request, when an exact match does not exist.

IX. CONCLUSION AND FUTURE WORK

In this paper we propose, develop and implement a service discovery architecture for ubiquitous computing environments. Our approach transforms BPEL user request and service descriptions into graphs which are semantically compared to produce a ranked list of services. We also limit the search space of potential services by initially matching of the user's request with those services which have been invoked previously by the current or other user with similar interests. Additionally, our approach filters the services which cannot be consumed on the user's device by comparing the user's device capabilities with the requirements of the service.

We have implemented our system as a prototype and presented an evaluation which assesses both the efficiency and accuracy of our approach. The evaluation shows that our approach is more efficient that using semantic reasoners providing **good** efficiency, performing 144 comparisons in under 1 second. We hypothesize that our approach provides acceptable efficiency for up to 4145.8 node comparisons, where acceptable implies a result was obtained within 10 seconds. U-ServiceMatch also provided extremely accurate results in terms of precision, achieving a result of 0.78-1. In terms of recall, a result of 0.7 or above was achieved with a semantic similarity threshold of 4 or above.

The next step of this work is to study and define new features that extend the user description in a ubiquitous environment. Additionally, we wish to implement a system of service registry, to reduce the search space where the *Service Repository* of considerable size in order to further improves efficiency.

REFERENCES

[1] J. Veijalainen, "Mobile ontologies: Concept, development, usage, and business potential," *International Journal on Semantic Web and Information Systems, Special Issue on Mobile Services and Ontologies*, vol. 4, pp. 20–34 2008.

[2] F. Almenárez, "Arquitectura de Seguridad para Entornos de Computación Ubicua Abiertos y Dinámicos," Tesis Doctoral. Departamento de Ingeniería Telemática, Escuela Politécnica Superior, 2005.

[3] J. Zoric, N. Gjermundshaug, and S. Alapnes, "Service mobility a challenge for semantic support," presented at the 16th IST Mobile and Wireless Communications Summit, IEEE, Budapest, Hungary, pp.1-7, 2007.

[4] C. Xiaosu and L. Jian, "Build mobile services on service oriented structure," in *IEEE International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1472–1476, 2005.

[5] B. Kargin and N. Basoglu "Factors affecting the adoption of mobile services," *Portland International Center for Management of Engineering and Technology, IEEE*, pp. 2993–3001, 2007.

[6] V. Roto and A. Oulasvirta., "Need for non-visual feedback with long response times in mobile hci," *International World Wide Web Conference Committee (IW3C2), Chiba, Japan*, pp. 775-781, 2005.

[7] S. Ben Mokhtar, *Semantic Middleware for Service-Oriented Pervasive Computing*. Tesis Doctoral, 2007.

[8] M. Sellami, S. Tata, and B. Defude, "Service Discovery in Ubiquitous Environments: Approaches and Requirement for Context-Awareness," ed Milan, Italy: BPM Workshops, pp. 516-522, 2009.

[9] T. Andrews, *et al.*, (2003, 05 05). Business Process Execution Language Version 1.1. the BPEL4WS Specification. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

[10] V. Hermida, O. Caicedo, J.C. Corrales, D. Grigori, and M. Bouzeghoub, "Service Composition Platform for Ubiquitous Environments Based on Service and Context Matchmaking", In the

4th Colombian Congress of Computer, Bucaramanga, Colombia, 2009.

[11] A. Bernstein and M. Klein, "Discovering services: Towards highprecision service retrieval," *International Workshop on Web Services, EBusiness, and the Semantic Web (CAiSE '02), Springer-Verlag, Toronto, Canada*, vol. 2512, pp. 260 – 275, 2002.

[12] W. Abramowicz, K. Haniewicz, M. Kaczmarek, and D. Zyskowski, "E-marketplace for semantic web services," *6th International Conference on Service-Oriented Computing (ICSOC '08), Springer-Verlag, Sydney, Australia*, vol. 5364, pp. 271 – 285, 2008.

[13] L. Ordoñez, A. Bastidas, C. Figueroa, and J.C. Corrales, "Task Semantic Comparison between the Telecommunications Business Processes," in *The 5th National Seminar on Emerging Technologies in Telecommunications and Telematics - TET*, Popayán, Colombia, 2010, pp. 26-31.

[14] V. Zacharias, *et al.*, "Mind the web," in *1st Workshop on New forms of Reasoning for the Semantic Web: Scalable, Tolerant and Dynamic in-conjunction with International Semantic Web Conference (ISWC '07) and Asian Semantic Web Conference (ASWC '07)*, vol. 291, CEUR-WS.org, Busan, Korea, 2007. Available: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-291/paper08.pdf.

[15] U. K. a. B. Köning-Rics, "Dynamic Binding for BPEL Processes - a Lightweight Aprproach to Integrate Semantics into Web Services," in *ICSQC*, 2007, pp. 116-127.

[16] H. Almohamed, "A linear programming approach for the weighted graph matching problem," presented at the IEEE Trans. PAMI 15, 1993.

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

[18] A. Bandara, *et al.*, "A Semantic Approach for Service Matching in Pervasive Environments," ed: Universidad de Southampton, 2007.

[19] J. C. Corrales, D. Grigori, M. Bouzeghoub, and J.E. Burbano, "Bematch: A platform for matchmaking service behavior models," *In the 11th International Conference on Extending database technology: Advances in database technology (EDBT'08)*, pp. 695-699, 2008, doi: 10.1145/1353343.1353428.

[20] W. Kokash, *et al.*, "Leveraging web services discovery with customizable hybrid matching," presented at the In Proc. of ICSOC, 2006, pp.522-528.

[21] E. Stroulia and Y. Wang, "Structural and semantic matching for assessing web-service similarity," presented at the Int. J. Cooperative Inf. Syst., 2005, pp. 407–438.

[22] L. Steller and S. Krishnaswamy, "Efficient Mobile Reasoning for Pervasive Discovery," in *Proceedings of the 2009 ACM symposium on Applied Computing (SAC 2009)*, pp. 1247-1251, 2009.

[23] J. Mendling, and J. Ziemann, "Transformation of BPEL Processes to EPCs", EPK 2005, Hamburg, Germany, vol. 167, December 2005, pp. 41-53.

[24] E. Guerrero, J.C. Corrales, and R. Ruggia, " Service Selection based on Profile Context and QoS Metamodels", in *The 5th Conference of the Euro-American Association on Telematics and Information Systems (EATIS'10)*, 2010. ISBN 978-958-44-7280-9, in press.

[25] S.P. Tocarruncho, F.A. Aponte, and A. Tocarruncho, "Extracción de Perfiles Basada en Agrupamiento Genetico para Recomendación de Contenido," in *Conferência IADIS Ibero-Americana WWW/Internet*, pp. 299-303, 2007.

[26] M. Zhang and N. Hurley, "Novel Item Recommendation by User Profile Partitioning," in *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, Milan, Italy, 2009, pp. 508-515.

[27] S. Abbar, *et al.*, "A personalized access model: concepts and services for content delivery platforms," in *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, Linz, Austria, 2008, pp. 41-47.

[28] S. Kurkovsky, V. Zanev, and A. Kurkovsky. "SMMART, a context-aware mobile marketing application: Experiences and lessons", Embedded and Ubiquitous Computing, vol. 3823, Springer-Verlag, Nagasaki, Japan, 2005, pp. 141 – 150.

[29] J D. Grigori, J.C. Corrales, M. Bouzeghoub, and A. Gater, "Ranking BPEL Processes for Service Discovery." vol. 3, ed: IEEE Transactions on Services Computing, pp. 178-192., 2010.

[30] G. Miller, "Wordnet: A lexical database for english.," in *Communications of the ACM*, vol. 38 no. 11, pp. 39–41, 1995.

[31] C. Doulkeridis and N. Loutas, M. Vazirgiannis. "A System Architecture for Context-Aware Service Discovery". International Workshop on Context for Web Services (CWS'05), Paris, France, pp. 101 - 106, 2006.

[32] Mobile W3C Device Independence Working Group, "Composite Capability / Preference Profiles (CC/PP): Structure and Vocabularies 2,0," ed, 2006.

[33] J. Vanhatalo, J. Koehler, and F. Leymann, "Repository for business processes and arbitrary associated metadata," presented at the BPM Demo Session at the Fourth International Conference on Business Process Management, Viena. Austria, 2006, pp. 25–31.

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

[35] D. Lewis, "Representation and learning in information retrieval," in *Ph.D. Thesis*, ed University of Massachusetts: Department of Computer and Information Science, 1992.

[36] L.J. Suarez, L.A. Rojas, J.C. Corrales, and O.M. Caicedo,(2010) Be4SeD: Benchmarking for evaluation of Service discovery techniques. *Revista de Ingeniería y Competitividad, Universidad del Valle*, in press.

[37] A. Duke, J. Davies, M. Richardson, and N. Kings, "A Semantic Service Orientated Architecture for the Telecommunications Industry ", INTELCOM, vol. 3283, 2004, pp. 236-245, doi: 10.1007/978-3-540-30179-0_21.

[38] WSML. *Web Service Modeling Language*. Available: http://www.wsmo.org/wsml/index.html

[39] WSML2Reasoner. *WSML2Reasoner framework*. Available: http://tools.sti-innsbruck.at/wsml2reasoner/