Proactive Service Discovery in Pervasive Environments

Fei Li[†], Katharina Rasch[‡], Hong-Linh Truong[†], Rassul Ayani[‡], Schahram Dustdar[†]

[†]Distributed Systems Group,
Vienna University of Technology
{li,truong,dustdar}@infosys.tuwien.ac.at

[‡]School of Information and Communication Technology (ICT),
Royal Institute of Technology (KTH)
{krasch,ayani}@kth.se

ABSTRACT

Pervasive environments are characterized by rich and dynamic context, where users need to be continuously informed about services relevant to their current context. Implicit discovery requests, triggered by changes of user context, available services, or user preferences are prevalent in such environments.

This paper proposes a proactive service discovery approach for pervasive environments to address these implicit requests. Services and user preferences are described by a formal context model, which effectively captures the dynamics of context and the relationship between services and users. Based on the model, we propose a proactive discovery algorithm to continuously present the most relevant services to the user in response to changes of context, services or user preferences. Numeric coding methods are applied in different phases of the algorithm to improve its performance. A proactive service discovery system is proposed and the context model is grounded in a smart home environment. Experimental results show that our approach can efficiently provide the user with up-to-date information about useful services.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Theory

1. INTRODUCTION

Pervasive environments are typically user-centric, featuring an increasing number of devices, rich user context and various user preferences. In such environments, Service-Oriented Architecture (SOA) has been widely applied for integrating devices, sensors, and software applications [3] [17].

ICPS2010 July 13-15, 2010, Berlin, Germany

This paper addresses a fundamental requirement in pervasive environments — to continuously discover the most relevant services for users in an ever-changing context.

Service discovery approaches in pervasive environments evolved from the traditional SOA field, where explicit user requests are the driving factors of service discovery. In pervasive environments, however, user context and user preferences become essential aspects when deciding which of the available services are most interesting to the user in a certain situation. These continuously changing aspects post a significant challenge to state-of-the-art service discovery mechanisms. Location changes have prevalently been addressed in the literature [15][9]. But other than location, the changes of time, environment, physical status, and service status can all affect the service discovery result. We argue that *most of the service discovery requests are implicit*, meaning that the system should discover services in response to the aforementioned changes, even if users did not issue an explicit request to the service system via a user interface. The capability of capturing these implicit requests will improve user experience significantly.

This paper presents a novel service discovery approach for pervasive environments. The goal is to proactively and continuously discover services that fit the ever-changing context and preferences of users. The discovery approach is based on a formal context model exploiting multi-dimensional space– *Hyperspace Analogue to Context (HAC)*. The model extends the concept of space beyond the spatial relationship commonly observed in the literature, so context properties other than location receive equal recognition when discovering services. Services are situated in HAC by formalizing their relationship to context. User preferences are modeled as scopes in the hyperspace, transforming service discovery to scope matching problem. We apply a series of numeric coding schemes to HAC for improving the performance of the discovery algorithm. Based on the proposed models and algorithm, we ground the HAC model in a smart home environment and present a proactive, context-driven service discovery system. Our experimental results show that our approach can provide users with continuous updates of relevant services in real-time.

The paper is organized as follows: Section 2 introduces a motivating scenario. Section 3 defines the Hyperspace Analogue to Context and related concepts. Section 4 proposes our proactive discovery algorithm and the service discovery system, followed by Section 5 which presents experimental

results of the algorithm. Section 6 surveys closely related work of service discovery and context modeling. The paper concludes with a discussion of future work.

2. MOTIVATING SCENARIO

Frida suffers from a neurodegenerative disease (Amyotrophic Lateral Sclerosis (ALS). She relies on a Brain Computer Interface (BCI) [6] for a number of daily activities like home appliance control, communication and handling emergencies. Nowadays BCI systems can recognize up to 50 different input commands [6]. However, to enhance usability, the number of alternatives displayed on the BCI input screen is often limited to less than 25 icons. This principle also applies more generally to mobile devices and all kinds of users: with too many input alternatives, user interfaces often get confusing and the usability decreases.

Several hundreds of services may be installed in a smart home environment, however only a fraction of those services are useful given a user's current context. A proactive service discovery system is needed, which can exploit dynamic user context, user preferences and service availability with the aim of identifying the best services for a given situation. Consider a typical morning with Frida awaking in her bed. Based on her current context and her preferences, the system displays different options on her BCI screen, for example *Raise headboard*, *Turn on TV* and *Open blinds*. After she has selected to adjust the headboard to a comfortable position, the system detects the context change and shows instead *Call nurse*. When she moves to the kitchen, bedroom services are no longer useful to her; her device could be updated with new options: *Make coffee*, *Check fridge content* and *Find recipe*. Meanwhile, a heavy storm is coming. A service to close all open windows in her house appears on her screen, *Close windows*. When having breakfast, Frida suddenly experiences difficulties swallowing, so via the BCI she notifies the system of this symptom by choosing *Report a health problem*, which also results in a context change. Such indications about Fridas dangerous physical status could also be reported by a sensor. A new service list containing *Call nurse*, *Call emergency service*, *Call relatives* is presented to Frida. All updates of the service list happen automatically without Frida explicitly requesting a new service discovery.

The scenario is only a fragment of Frida's daily life. Her context is rich and ever-changing, which includes temporal, spatial, device information, user actions and user status. It is simply impractical for the user to keep track of all context changes and issue explicit discovery requests accordingly. A service discovery approach is not only necessary to identify all available services in a given context, but also to select the ones of most interest to the user. For dealing with the different types of context information, we need to define a formal model for describing context, services as well as user preferences. Based on this model we can then propose a proactive discovery algorithm, which reacts to context changes on-the-fly, identifies fitting services and matches those with the user's preferences. Although our scenario is based on a special-needs user, the expected capabilities are appealing to all users.

3. HYPERSPACE ANALOGUE TO CONTEXT

Hyperspace Analogue to Context (HAC) is a concept to model context as a multi-dimensional space, effectively capturing continuously changing information from various context sources. In this section we describe HAC and its operations in a series of definitions. Based on these, we can model all necessary information for our proactive service discovery algorithm, including services and user preferences.

3.1 Basic definitions

DEFINITION 1 (N-DIMENSIONAL HAC). *An n-Dimensional HAC is a space* $\mathbb{H} =< \mathbb{D}_1, \mathbb{D}_2 ... \mathbb{D}_n >$, *where each dimension* $\mathbb{D}_i$ *denotes a type of context.*

In HAC, a dimension is the meta data to describe the data type and value set for a specific type of context, for example location, time, and status of a service. Depending on the data type, the values of a dimension can be continuous or discrete, infinite or limited. For a location dimension, the values could be the rooms in a house; for a temperature dimension it could be the values between zero and one hundred degree Celsius. All dimensions together span the n-dimensional space of all potential context descriptions. The number of dimensions may be large because of the complexity of pervasive environments; however most context descriptions will use only a fraction of all dimensions.

DEFINITION 2 (CONTEXT POINT). *The context point of an object o in space* $\mathbb{H}$ *is* $c^o =< d_1, d_2 ... d_n >$, *where* $d_i \in \mathbb{D}_i$.

The context of an object is described as a point in HAC. The changing of its context is considered as the object moving in HAC. For example, in our scenario the context of Frida may be $< d_{location} = bedroom, d_{physical} = normal >$. When an emergency happens, Frida is moved to another point: $< d_{location} = bedroom, d_{physical} = abnormal >$.

DEFINITION 3 (CONTEXT SCOPE). *A context scope C is a subspace in* $\mathbb{H}$. $C =< D_1, D_2 ... D_n >$, *where* $D_i \subseteq \mathbb{D}_i$.

A context scope limits the value sets for the dimensions. It is often used to describe a condition, e.g.$< D_{humid} = [60 \dots 70], D_{temp} = [15 \dots 18] >$ describes a condition for the temperature to be between 15 and $18°C$ and the humidity between 60% and 70%. We define a context point c to be within a context scope C as: $c \in C \iff \forall i, d_i \in D_i$.

3.2 Operations in HAC

DEFINITION 4 (BASIS). *In an n-Dimensional HAC, a basis is a vector* $B =< b_1, b_2 ... b_n >$, *where* $b_i \in \{0, 1\}$.

The basis identifies those context dimensions relevant for a context description. The basis is defined for both context scope and context point. The basis of a context scope C is $B(C) =< b_1, b_2 ... b_n >$, where $b_i = 0 \iff D_i = \phi$. The basis of a context point c is $B(c) =< b_1, b_2 ... b_n >$, where $b_i = 0 \iff d_i$ is not relevant. For example, in $\mathbb{H} =< \mathbb{D}_1, \mathbb{D}_2, \mathbb{D}_3 >$ a context point $c =< d_1, d_3 >$ has the basis $B(c) =< 1, 0, 1 >$.

DEFINITION 5 ($\times B$). $\times B$ *is an operation to render a partial view of context.* $C' = C \times B$, *when* $\forall i, (D'_i = D_i \iff b_i = 1) \wedge (D'_i = \phi \iff b_i = 0)$.

This operation can be applied to both context scope and context point. For example, if $\mathbb{H} =< \mathbb{D}_1, \mathbb{D}_2, \mathbb{D}_3 >$ with $c =< d_1, d_2, d_3 >$ and $C =< D_2 >$, then $c \times B(C) =< d_2 >$

DEFINITION 6 ($\times\Delta c$). *$\times\Delta c =< \Delta d_1, \Delta d_2 ... \Delta d_n >$ is an operation to change a context point. Δd_i is the difference of a certain dimension, $d'_i = d_i \times \Delta d_i$. If d_i doesn't change, $\Delta d_i = \phi$.*

The Δc operation effectively moves a context from one point to another in HAC. Context changes as described by Δc are the main driving force of service discovery in HAC.

3.3 Services and user preferences

DEFINITION 7 (CONTEXT-AWARE SERVICE). *A context-aware service s is situated in HAC. It can be invoked in a certain context scope and invoking it will change the context of the user. Thus, two types of context scopes characterize the relationship between services and HAC.*

- *Input context C^{sI} is the triggering condition of service s. When user is in C^{sI}, service s becomes one possible choice. Formally, $c^u \times B(C^{sI}) \in C^{sI}$.*
- *Output context C^{sO} is the possible context after running a service. Formally $c^u \times B(C^{sO}) \in C^{sO}$.*

If a service s is successfully executed, a transition of user context $c^u \rightarrow c'^u$ happens such that $c^u \times B(C^{sI}) \in C^{sI}$ and $c'^u \times B(C^{sO}) \in C^{sO}$. An example service s which turns on the oven and heats it up to a desired temperature could be described as having input context $C^{sI} =< D_{oven} = [off] >$ and $C^{sO} =< D_{oven} = [on], D_{ovenTemp} = [120 \dots 250] >$.

According to Definition 4, the basis of the input context of each service is a n-dimensional vector, indicating which dimensions are relevant to the service. This concept is one of the key factors in our algorithms performance.

DEFINITION 8 (USER PREFERENCES). *The preferences of user u are defined as the set of context scopes that the user would like to be situated in. $\mathbb{P}^u = \{(w_1, P_1), (w_2, P_2) \dots, (w_t, P_t)\}$, where each P_i is a context scope, $w_i \in (0, 1)$ is the weight of each preference.*

User preferences describe the goal of service discovery: suggesting relevant services that lead to a new context to match a preferred context given by a user. A user will typically have many preferences for different situations. A preference definition P_i may set preferred values for one or more dimensions. We use the notion of context scope rather than context point for preference because a scope is more flexible for expressing the possibly fuzzy goals of the user, such as "the temperature should be between 20-25°C". The weight represents the importance of each preference, e.g. a "no fire" preference is obviously more important than one concerning a comfortable lighting. It needs to be noted, that the collection and analysis of user preference is beyond the scope of this paper.

3.4 HAC in the smart home

There is a plethora of context types [1][20] that can form the dimensions for HAC. In the following we introduce a list of dimensions we have identified in typical smart home environments. However the list is intended not to be exhaustive, but illustrative. The dimensions can easily be adapted or extended to more general pervasive environments.

Location The location $\mathbb{D}_{location}$ of persons or objects is a context dimension typically used in context-aware systems. The area that a service is available in is of major importance for determining which services are applicable in a given situation. We mainly use relative location between objects rather than absolute coordinates because they are more convenient for service discovery and more intuitive to user. e.g. *in the kitchen* or *in front of the house*.

Time The time $\mathbb{D}_{time}$ describes when an action is happening. In some services a user may be interested only within a specific time frame. The time can be described in absolute terms (*November 19th 2009, 11:01 am*) or relative terms (*after the washing is finished*).

Environment Service discovery can also be driven by changes in the environment. In our scenario, the service *Close windows* is presented after the detection of rain. Each relevant environmental property can be seen as a context dimension. In a smart home we may for example monitor the temperature $\mathbb{D}_{temperature}$, the humidity $\mathbb{D}_{humidity}$ or the noise level $\mathbb{D}_{volume}$.

Physical status Physical status is critical to users in need of continuous monitoring and caring, such as Frida in our scenario. The heart rate $\mathbb{D}_{heartrate}$, breath rate $\mathbb{D}_{breathrate}$, and blood pressure $\mathbb{D}_{bloodpressure}$ are the general metrics. More specific dimensions in this category can be added for specific diseases and with the support of special devices.

Device status The status of devices has direct effects on the service discovery result. Obviously the service *Close windows* is of no use if the windows are already closed. In the smart home we can identify a multitude of potential device statuses as dimensions, for example for media devices $\mathbb{D}_{mediaStatus}$ *(on, off, play, pause, forward)* or the coffee machine $\mathbb{D}_{coffeeStatus}$ *(on, off, making coffee)*. Quality of Service (QoS)[16][14] metrics can also be described as dimensions of device status.

4. PROACTIVE SERVICE DISCOVERY

4.1 Context matching

For the service discovery algorithm we need to be able to assess how well a service can fulfill a user preference, i.e. whether the service output context changes the current context in such a way that the user preference is fulfilled. For comparing services and selecting the ones that match a given set of user preferences best, a numerical representation of how well one context scope matches another one is necessary.

4.1.1 Context matching score

Evaluating how well a service matches a request is a well-known problem in web service discovery research. Typically the subsumption hierarchy of service parameters and capabilities is used to find out which services can fulfill a request [13]. In context matching we are not only looking for full matches, i.e. a service output context that can fully fulfill a preference. Information about which services bring the system nearer to the user goal is just as valuable. Intuitively we need to calculate how much of the user preference context is covered by the service output context.

The context matching score $matching(C^1, C^2)$ is a numerical assessment of how well C^2 fulfills C^1. It is composed of the individual matching of dimensions in the scopes. Since scope C^1 is to be matched, only the k dimensions set in C^1, i.e. those with $B(D) = 1$, need to be considered. According to Equation 1, the dimensional matching value $dmatch(D^1, D^2)$ is determined by calculating the overlap between the two dimension values D^1 and D^2, dividing it by the size of D^1. This equation captures the notion that if D^2 is fully contained in D^1, then $dmatch(D^1, D^2) = 1$. If $B(D^1) = 0$ or $B(D^2) = 0$, of course $dmatch(D^1, D^2) = 0$.

$$dmatch(D^1, D^2) = \begin{cases} \frac{|D^1 \cap D^2|}{|D^1|} & \text{if } B(D^1) = 1 \wedge B(D^2) = 1 \\ 0 & \text{else} \end{cases} \quad (1)$$

The overall matching score is determined according to Equation 2 by adding the individual dimension matching scores and dividing by k, the number of set dimensions in C^1. Dividing by k effectively penalizes the dimensions that can not be fulfilled by C^2.

$$Matching(\mathbb{C}^1, \mathbb{C}^2) = \frac{\sum_{i=1}^{n} dmatch(D_i^1, D_i^2)}{k} \quad (2)$$

4.1.2 Numerical encoding

The actual calculation of the dimension overlap depends of course on the datatype of the respective dimensions. For numerical values on an interval scale, this is straightforward. However in a smart home there are many dimensions with non-numerical values. This includes the status of various devices, e.g. play/pause/start/on/off for a media center or a location expressed as rooms of the house. In these cases it is more adequate to model a dimension with concepts in an ontology [5] than with numerical values. Figure 1 shows for example an extract of an ontology for the location dimension in a smart home. As in this example, the expressiveness of a taxonomy using only is-a relationships is typically sufficient when modeling context dimensions.

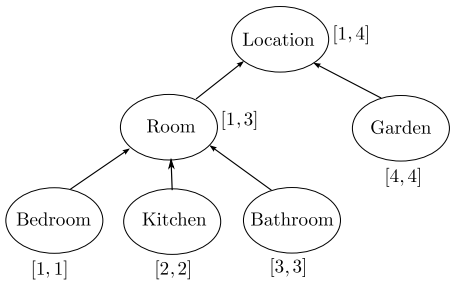

Figure 1: Taxonomy of location concepts

For being able to efficiently calculate the dimension matching of non-numeric dimensions, we encode the ontology using the postorder interval scheme proposed by Agrawak et al [2]. A concepts postorder number is defined as its position in a postorder (depth-first) traversal of the hierarchy. Each concept is represented by an interval of the form $[i, j]$, with j being the postorder number of the concept and i being the lowest postorder number among its descendants. The intervals reflect thereby the subsumption hierarchy, a concept B is subsumed by a concept A, if B's interval lies within A's interval. For a correct calculation of the overlap between concepts, we needed to slightly modify the encoding such that j is represented by the highest postorder number among the descendants of the nodes. Figure 1 shows the intervals for the concepts of the location ontology.

By using this encoding scheme, the dimension matching for ontology concepts becomes very similar to numerical dimension matching. Set operations are used to determine the number of concepts contained in a dimension value and in the overlap between two values. In the location ontology, for example, the overlap between $D^1 = \{room\}$ and $D^2 = \{kitchen\}$ can be calculated by $dmatch(D^1, D^2) = \frac{|\{1,2,3\} \cap \{2\}|}{|\{1,2,3\}|} = \frac{1}{3}$. The encoding scheme allows for a very efficient calculation of the dimension match value without the need for reasoning about the subsumption hierarchy. The modeling of a context dimension can be considered static, so that the encoding of the ontology can be pre-computed at configuration time.

4.2 Proactive service discovery algorithm

Based on the previous introductions of matching algorithm and ontology coding scheme, this section presents our efficient proactive algorithm for service discovery in HAC. The efficiency of the algorithm is assured by two mechanisms. First, services that are not affected by a context change are filtered out with a fast bit-set operation and excluded from the evaluation phase. Second, we maximize the reuse of previous discovery results by keeping those services not affected by the changed context in the result set. Algorithm 1 runs continuously in response to the update of context information. We call each execution of the algorithm a round. A new round can also be started by adding new services or changing user preferences.

Figure 2 illustrates the relationship between algorithm input and output. The input parameters of Algorithm 1 are as follows.

- S_{pre} is the service set discovered in the previous round of the algorithm. S_{pre} contains the services currently presented to the user. For the initial round of the algorithm, S_{pre} is set to empty, ϕ. It will then be updated iteratively with the result of each execution round, S_{ranked}.

- S is the whole set of services registered in the environment. The service input and output context are represented as numeric values according to the concept coding mechanisms presented in Section 4.1.

- c^u is the current context of the user. c^u is a context point with the dimension values set to reflect the current status of the user. Again all elements of the context are represented by numeric values.

- Δc^u is the change of user context compared to the previous status of c^u, as defined in Definition 6. In our algorithm, only the basis of Δc^u is used, i.e. only the bit-set telling the algorithm which dimensions have changed.

- $\mathbb{P}^u$ is the set of user preference (Definition 8).

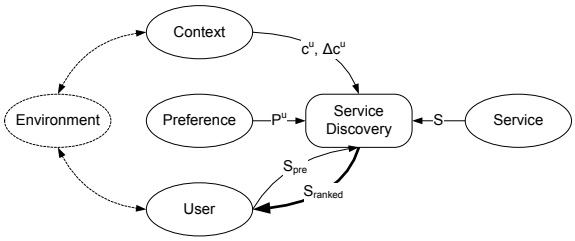

Figure 2: Illustration of algorithm input and output

Algorithm 1 Proactive Service Discovery

```
1: procedure ServiceDiscovery(S_pre, S, c^u, Δc^u, P^u)
2:     S_cand = φ
3:     for ∀s ∈ S do
4:         if B(Δc^u) ∧ B(C^sI) ≠ 0 then
5:             S_cand = S_cand + s
6:         end if
7:     end for
8:     S_ranked = S_pre − S_cand
9:     for ∀s ∈ S_cand do
10:        c'^u = c^u × B(C^sI)
11:        if c'^u ∈ C^sI then
12:            s.score = 0
13:            for ∀(w_i, P_i) ∈ P^u do
14:                C^sO' = C^sO × B(P^u)
15:                score = w_i * Matching(P_i, C^sO')
16:                if score > s.score then
17:                    s.score = score
18:                end if
19:            end for
20:            S_ranked.InsertOrdered(s)
21:        end if
22:    end for
23:    return S_ranked
24: end procedure
```

Lines 3 to 7 form the first phase of our service discovery, which identifies the services that are affected by the context change and adds them to a candidate service set S_{cand}. The basis of the context change and the service input context is evaluated: $B(\Delta c^u) \wedge B(C^{sI}) \neq 0$. A non-zero result indicates that service s is listening to at least one of the changed dimensions in Δc. Although the loop is applied to all the services in the current environment, it costs only a very small fraction of the algorithm execution time, since for each service it simply entails a fast bit-set operation. S_{cand} is the input to the second phase of discovery.

Line 8 initializes the result of the algorithm S_{ranked}. The result of the previous round is reused by keeping all previously discovered services that are not affected by the context change. The intersection of S_{pre} and S_{cand}, however, contains those previously discovered services that are affected by the changed context and that will therefore have to be re-evaluated in the second phase of the algorithm.

Candidate services are ranked and evaluated in lines 9-22. Line 10 keeps in c'^u only the dimensions related to the input context of a service. If c'^u is in the scope of service input context C^{sI} (line 11), this service will be included in the algorithm result.

The next steps evaluate how well the service fulfills the given user preferences. Again, only those dimensions of the service output context are considered, that are relevant to the user preference (line 14). For each of the preferences, the matching method is invoked in line 15. The matching score is tuned by the weight of each preference. For each service only the highest score of preference matching will be used for ranking (line 16-17). Recalculating the score each time is unnecessary for the currently static preferences and service descriptions, but will become important in our future work where we plan to support also dynamic user preferences. The result is inserted into the ranked service set according to the matching score. Figure 3 illustrates the ranking phase.

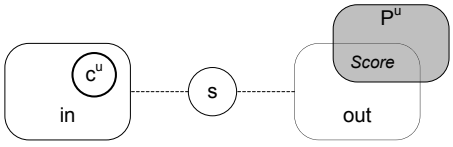

Figure 3: Illustration of ranking

4.3 System architecture

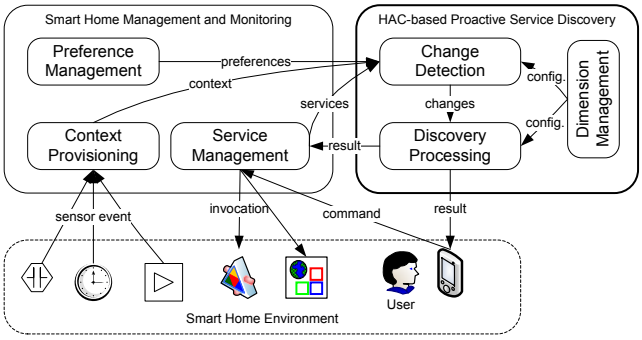

Figure 4: System architecture

Figure 4 depicts how the HAC-based proactive service discovery interacts with other components in our smart home system. A typical smart home environment contains sensors to detect the context and services to be invoked by users. Users interact with the system via user devices, e.g. BCI and mobile devices. Context information from heterogeneous sources is collected and processed by the *Context Provisioning* component, which is a complex event processing system, producing formatted context events. The *Service Management* stores service descriptions, monitors and invokes home services. Service registration and unregistration are also performed by this component. The *Change Detection* decides when to trigger service discovery. It aggregates context, preference and service information as input parameters of the service discovery algorithm. The threshold for invoking the discovery algorithm depends on each context dimension. Especially for numeric dimensions, the threshold that constitutes a context change should be decided by user requirements and system performance. For example, it is unnecessary to rediscover services for every temperature sensor reading. Our algorithm is implemented in the *Discovery Processing* component. Since HAC is a general concept and specific smart-home service environments can

have different sets of dimensions depending on the available devices and user requirements, the *Dimension Management* component is introduced to customize the types of context information. Numerical encoding is carried out by this component when adding or changing a dimension. It is used for configuring not only our service discovery approach for dealing with diverse context information, but also the change detection component to interface with different context provisioning and preference management systems.

5. EXPERIMENTS

5.1 Performance evaluation

The performance of the service discovery significantly influences the users experience. Context changes can happen very frequently and the discovery results should quickly reflect these changes. We have identified five important variables that can influence the performance of the algorithm: *(i)* the number of available services, *(ii)* the number of dimensions, *(iii)* the number of user preferences, *(iv)* the percentage of services affected by a context change (in the following referred to as affected services), and *(v)* the percentage of services whose matching score needs to be calculated (in the following referred to as applicable services).

We have performed several experiments to show the system performance in different settings. For each of the experiments we have generated testing sets that vary some of the variables, keeping the rest fixed. We have used conservative estimates of a real smart home when setting the experimental input, e.g. we generally assume that 30% of all services are applicable — in a real home with a high service diversity we expect a much smaller number of applicable services, which will in turn result in shorter execution time of our algorithm. All of the experiments were performed on a desktop PC with an Intel Core 2 Duo CPU with 3 GHz and 4 GB RAM running Linux, and were run for 100 replications.

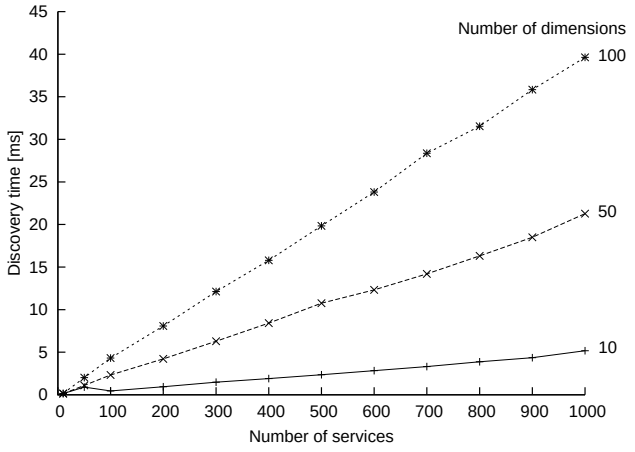

Figure 5: Discovery time depending on number of services and dimensions (30% affected services, 30% applicable services, 25 preferences)

Figure 5 shows how the discovery time depends on the number of services and dimensions. It can be seen that the system scales well for the number of dimensions and services. With 1000 services and 100 dimensions, the discovery time equals to circa 40 milliseconds (with a standard deviation of 1.3 milliseconds), allowing for a near real-time update of the user interface.

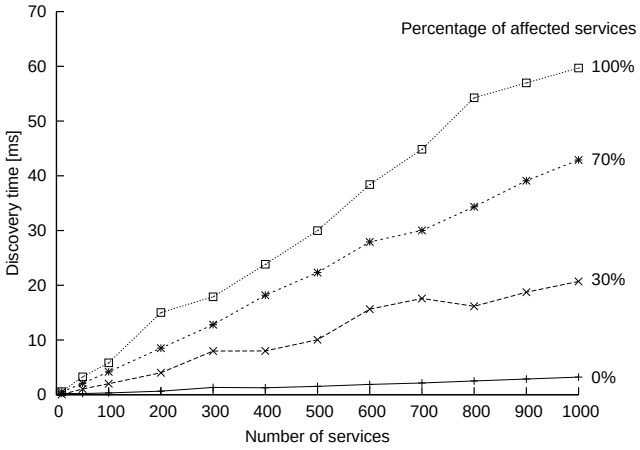

Figure 6: Discovery time depending on number of services and percentage of affected services (50 dimensions, 30% applicable services, 25 preferences)

Figure 6 shows how the discovery time depends on the percentage of affected services. The best results can of course be achieved with 0% affected services; in this case only the bit-set operations to identify the services affected by the context change need to be performed. When increasing the number of affected services, also the number of necessary matching score calculations increases. We can see that the system also scales well in this regard, for 100% affected services, the discovery time increases by circa a tenfold, equaling circa 60 milliseconds (with a standard deviation of 1.4 milliseconds).

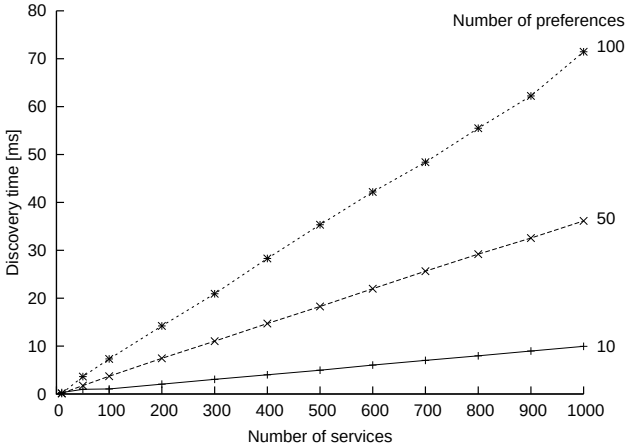

Figure 7: Discovery time depending on number of services and preferences (50 dimensions, 30% applicable services, 30% affected services)

The influence of the percentage of affected services and the number of preferences both concern the matching part

of the algorithm. We could therefore safely measure just the influence of the preferences, and from the results draw conclusions also about the influence of the percentage of affected services. The results of this experiment can be seen in Figure 7. Again even with a tenfold increase of services as well as preferences, the discovery time only increases by a tenfold (70 milliseconds, with a standard deviation of 1.8 milliseconds).

5.2 Effectiveness evaluation

We have tested how well our discovery approach is able to present the most relevant services to the user with only a limited number of icons on the user interface. We first generated a set of 10 preferences and of 100 applicable services such that each service fulfills one of the preferences to a randomly varying degree. We then ran the service discovery algorithm and trimmed the results down to the k best services, with k being the number of icons to be displayed. To find out how well this service selection fulfills the preferences, we have merged all preferences into one context scope $PrefAll =< D_1, D_2...D_n >$, where D_i is defined by Equation 3, in which t is the number of preferences. Similarly the output context of all services is merged to *ServAll* and that of the k selected services to *ServSel*. We could then calculate how well the preferences are covered using Equation 4, which normalizes the matching score between *PrefAll* and *ServSel* according to the maximum matching score achievable with all services. For coverage calculation, we assumed the same weight for all preferences.

$$D_i = \bigcup_{j=1}^{t} D_i^{P_j} \tag{3}$$

$$Coverage = \frac{Matching(PrefAll, ServSel)}{Matching(PrefAll, ServAll)} \tag{4}$$

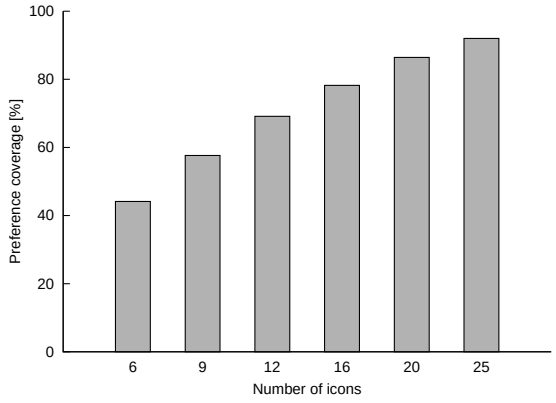

Figure 8: Preference coverage depending on the number of icons displayed

Figure 8 shows the results of this experiment for a varying number of icons. Even with only service 9 icons, we can already achieve around 60% of the maximum preference coverage, with 25 icons over 90% can be achieved. The remaining 75 services not included in this selection fulfill the preferences only to a very small degree, so our discovery result is an optimal tradeoff between the limitation of user interface and desirable services.

6. RELATED WORK

Service discovery in pervasive environments has been intensively investigated in recent years [19][18]. Early discovery approaches were based on the functional description of services, for instance, service category, semantic description and key words. For enhancing service discovery, many research efforts involved Quality of Service (QoS) and context in different phases. However, as far as we know, few work has acknowledged context as a first-class criterion and motivating factor in service discovery. *In other words, explicit, request-driven service discovery approaches are predominant in pervasive environments.* Two examples are given in the following paragraph.

Mokhtar and Preuveneers et al. [11] proposed the EASY (Efficient semAntic Service discoverY) framework which takes QoS and context into account. User preference is not considered in EASY. To improve the performance of service discovery, the semantic description of services is optimized by numeric coding scheme, which is a widely adopted method to improving the performance of ontology processing. In our work, another set of numeric coding schemes are applied to the essential aspects of context model, i.e. basis and context concepts. Park and Yoon et al. [14] presented the concept of Virtual Personal Space (VPS) to extend the scope of service discovery. VPS conceptually extended the *concept of space* beyond the location domain by including QoS, user rating and service load in service discovery. User preference is not explicitly expressed, but inferred from feedbacks. The factors considered in VPS are service-oriented rather than context-oriented. Moreover, neither EASY nor VPS consider the impact of continuous context changes on service discovery.

There are few works that have addressed proactive discovery as a complement to explicit request-driven service discovery. However, our work distinguishes itself substantially. Bellavista and Corradi et al. [4] proposed a user-centric service view for explicit discovery. Similarly, Hesselman and Tokmakoff et al. [7] presented the idea of a persistent discovery request. Our goal is similar to these two in terms of updating the set of relevant services dynamically. However, we described a comprehensive formal model which contains different types of context, and an efficient numeric-based algorithm to reflect the changes in context, user preference, and services in real time. Moreover, detailed performance of our approach and the method for evaluating discovery results are presented.

Our HAC has been inspired by (HAL) Hyperspace Analogue to Language [10] and the context space of assumptions [8]. Both approaches originated in the Artificial Intelligence domain. HAL is used for understanding natural language and measuring the difference between statements. Context space of assumptions aims to analyze interpretation of assumptions within different communication contexts, which is completely different to the meaning of context in pervasive environments. In our domain, Padowitz and Loke et al. [12] have proposed a usage of space theory for situation reasoning. In this work concepts are generally treated as non-numeric enumerates so that no comparison is applicable, thereby leaving out a large spectrum of context information. In contrast, HAC uses ontologies and numeric coding to characterize the relationship between concepts. HAC also formalizes the methods to alter different views of dimensions by basis and to describe context transitions by

changes. Most importantly, our goal is largely different to situation reasoning, so based on HAC, we model services and user preferences to propose a context-driven service discovery approach.

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed a proactive service discovery solution for pervasive environments. In such environments, users need to have access to the most relevant and interested services within a rich and dynamic context. A theoretical model—Hyperspace Analogue to Context (HAC)—is proposed to describe context, services and user preference. HAC includes useful operations to tailor context views and describe context changes. Since performance is an important factor for the usability of continuous service discovery, we have applied a series of performance improvement approaches. Services that are related to a specific context update are identified with a fast bit-set operation. Context ontologies are encoded numerically so that no costly reasoning has to be performed during the discovery. And existing service discovery results are efficiently reused to minimize the need for context matching. Our experimental results proved that our system can efficiently and effectively provide users with up-to-date information about the most relevant and interesting services. We have proposed a set of example dimensions and a proactive service discovery system to embed the algorithm into a smart home environment. In addition, our context model and discovery approach could also assist traditional, explicit service discovery by limiting their search scope.

We are currently implementing the interface to BCI and mobile user devices. Field tests with different kinds of users will be conducted to get valuable feedback on our service discovery system. We also plan to improve the configurability and usability of proposed approach. We will incorporate an online feedback analysis that updates user preferences based on typical service choices in certain context. Furthermore, the mechanism to dynamically assign dimension weights in the service matching calculations is under investigation.

Acknowledgment

This work is supported by EU FP7 STREP Project SM4ALL (Smart hoMes for ALL), under Grant No. 224332.

8. REFERENCES

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *HUC*, pages 304–307, 1999.

[2] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. *SIGMOD Rec.*, 18(2):253–262, 1989.

[3] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad-Hoc and Ubiquitous Computing*, Jan 2006.

[4] P. Bellavista, A. Corradi, R. Montanari, and A. Toninelli. Context-aware semantic discovery for next generation mobile systems. *Communications Magazine, IEEE*, 44(9):62–71, 2006.

[5] T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[6] C. Guger, C. Holzner, C. Groenegress, G. Edlinger, and M. Slater. Brain-computer interface for virtual reality control. In *Proceedings of ESANN 2009*, pages 443–448, 2009.

[7] C. Hesselman, A. Tokmakoff, P. Pawar, S. Iacob, et al. Discovery and composition of services for context-aware systems. *Lecture Notes in Computer Science*, 4272:67, 2006.

[8] D. Lenat. The dimensions of Context-Space. 1998.

[9] S. W. Loke, S. Krishnaswamy, and T. T. Naing. Service domains for ambient services: concept and experimentation. *Mob. Netw. Appl.*, 10(4):395–404, 2005.

[10] K. Lund and C. Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods Instruments and Computers*, 28(2):203–208, 1996.

[11] S. B. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers. Easy: Efficient semantic service discovery in pervasive computing environments with qos and context support. *The Journal of Systems & Software*, 81(5):785âĂŞ808, 2008.

[12] A. Padovitz, S. Loke, and A. Zaslavsky. Towards a theory of context spaces. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 38–42, March 2004.

[13] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *The Semantic Web âĂŤ ISWC 2002*, volume 2342/2002, pages 333–347. Springer Berlin / Heidelberg, January 2002.

[14] K. Park, U. Yoon, and S. Kim. Personalized service discovery in ubiquitous computing environments. *Pervasive Computing, IEEE*, 8(1):58–65, 2009.

[15] M. Park, J. Hong, and S. Cho. Location-Based recommendation system using bayesian userâĂŹs preference model in mobile devices. In *Ubiquitous Intelligence and Computing*, pages 1130–1139. 2007.

[16] M. K. S. Cuddy and H. Lutfiyya. Context-aware service selection based on dynamic and static service attributes. In *Proceedings of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 13–20, 2005.

[17] H.-L. Truong and S. Dustdar. A survey on context-aware web service systems. *International Journal of Web Information Systems*, 5(1):5–31, 2009.

[18] C. Ververidis and G. Polyzos. Service discovery for mobile ad hoc networks: a survey of issues and techniques. *Communications Surveys & Tutorials, IEEE*, 10(3):30–45, 2008.

[19] F. Zhu, M. Mutka, and L. Ni. Service discovery in pervasive computing environments. *Pervasive Computing, IEEE*, 4(4):81–90, 2005.

[20] A. Zimmermann, A. Lorenz, and R. Oppermann. An operational definition of context. In *Modeling and Using Context*, pages 558–571. 2007.