World Wide Web
DOI 10.1007/s11280-011-0112-x

Context-driven personalized service discovery in pervasive environments

Katharina Rasch · Fei Li · Sanjin Sehic · Rassul Ayani · Schahram Dustdar

Received: 15 July 2010 / Revised: 5 November 2010 /
Accepted: 17 January 2011

Abstract Pervasive environments are characterized by a large number of embedded devices offering their services to the user. Which of the available services are of most interest to the user considerably depends on the user's current context. User context is often rich and very dynamic; making an explicit, user-driven discovery of services impractical. Users in such environments would instead like to be continuously informed about services relevant to them. Implicit discovery requests triggered by changes in the context are therefore prevalent. This paper proposes a proactive service discovery approach for pervasive environments addressing these implicit requests. Services and user preferences are described by a formal context model called Hyperspace Analogue to Context, which effectively captures the dynamics of

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

K. Rasch (✉) · R. Ayani
School of Information and Communication Technology (ICT SCS),
Royal Institute of Technology Stockholm (KTH), Forum 120, 16440 Kista, Sweden
e-mail: krasch@kth.se

R. Ayani
e-mail: ayani@kth.se

F. Li · S. Sehic · S. Dustdar
Distributed Systems Group, Vienna University of Technology,
Argentinierstrasse 8/184-1, 1040 Wien, Austria

F. Li
e-mail: li@infosys.tuwien.ac.at

S. Sehic
e-mail: sehic@infosys.tuwien.ac.at

S. Dustdar
e-mail: Dustdar@infosys.tuwien.ac.at

context and the relationship between services and context. Based on the model, we propose a set of algorithms that can continuously present the most relevant services to the user in response to changes of context, services or user preferences. Numeric coding methods are applied to improve the algorithms' performance. The algorithms are grounded in a context-driven service discovery system that automatically reacts to changes in the environment. New context sources and services can be dynamically integrated into the system. A client for smart phones continuously informs users about the discovery results. Experiments show, that the system can efficiently provide the user with continuous, up-to-date information about the most useful services in real time.

Keywords pervasive environments · context awareness · context model · service discovery

1 Introduction

Pervasive environments are user-centric, featuring an increasing number of devices, a rich user context and various user preferences. In such environments, Service-Oriented Architecture (SOA) has been widely applied for integrating devices, sensors, actuators and software applications [4, 23]. State of the art service discovery approaches in pervasive environments use techniques 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 of most interest to the user in a given situation. The user context is rich and ever-changing; it covers aspects such as user location, current time, environmental information and the users' health status. These continuously changing aspects pose a significant challenge to state of the art discovery mechanisms. We argue that most service discovery requests in pervasive environments are implicit. The system should discover services in response to changes in the user context, even if the user did not issue an explicit service discovery request to the system. We will show, that the capability of capturing these implicit requests can improve user experience significantly.

In this paper we propose a novel, proactive service discovery approach for pervasive environments. The discovery approach is based on a formal model of context 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 [16, 22], so context properties other than location receive equal recognition when discovering services. We use the concepts in HAC to model user preferences and the capabilities of services in relation to context. The key in service discovery becomes thus to identify changes in the user context and to filter out unsuitable services by checking their preconditions against the new context. Those services that are of most interest to the user can then be selected by matching their effects with the user's preferences.

The proactive discovery algorithm is the core of our service discovery system. The system integrates various types of sensors, devices and appliances and makes them available as context-aware resources. New context sources and services can easily be described using a set of Java annotations. They can be dynamically deployed to the system and are seamlessly integrated into the pervasive environment. Changes in the user context, the user's preferences and in the set of available services are automatically detected and the service discovery algorithm is initiated when necessary. Users can get continuous updates with the list of services via an interface on mobile phone devices. Our experimental results show, that our system can provide the user with continuous updates of interesting services in real time.

In Li and Rasch et al. [15] we have presented the problem for the first time and our preliminary solutions to it. Compared to our previous work, in this paper we have improved and extended our solution in several aspects. First, the discovery algorithm is further optimized with an offline service score calculation procedure and a change detection method based on sampling. Second, the service discovery system is implemented and integrated with a specialized context provisioning system. Third, we did more extensive evaluations of our prototype towards the end-to-end discovery time in more realistic scenarios.

The paper starts with a scenario motivating the need for a proactive, context-driven service discovery system. Section 3 re-introduces our Hyperspace Analogue to Context and related concepts. Section 4 presents an extended and improved set of algorithms for proactive service discovery. A smart home system based on HAC is described in Section 5. In Section 6 we evaluate both our formal model of context and the discovery algorithms as well as the performance of our system. Section 7 surveys closely related work.

2 Motivating scenarios

Frida Frida suffers from a neurodegenerative disease (Amyotrophic Lateral Sclerosis ALS). She relies on a Brain Computer Interface (BCI) [11] 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 [11]. However, to enhance usability, the number of alternatives displayed on the BCI input screen is often limited to less than 25 icons.

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 breathing; such indications about Frida's dangerous physical status are 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.

Alex Alex is a young technician who is open to new technologies. He uses the services in his smart home environment via an Android smart phone on which a client software is installed to continuously display the services most interesting to Alex depending on his current situation. His main interests in using a smart home system are information access and entertainment rather than controlling his household appliances. Alex often buys new devices and expects that he can integrate them into the smart home system with minimal effort.

Alex has had dinner and enters his living room. According to Alex' preferences, the smart home system suggests several choices for recreation: *Watch TV*, *Movie program* and *Play Games*. He opts to watch a movie in a local cinema. The local movie program is retrieved and he selects a movie that will be shown in one hour. When he has selected an entertainment service, the service list is updated with some other activities he typically does after dinner: *Light the fireplace* and *Call girlfriend*. He selects to call his girlfriend via internet phone; this prompts his smart home client to be updated with the service *Mute radio*. When Alex is about to leave for the movie, his system displays a set of services to ensure the correct status of the house for his absence, such as to turn off the lights and appliances and to lock the doors and windows. Again the client is updated by the smart home without explicit discovery requests.

The scenarios are only fragments of the daily life of two users. User context is rich and ever-changing, including temporal, spatial, device information, user actions and user status. From several hundreds of services that may be available in a smart home environment, only a small percentage will be of interest to the user in any given situation. It is simply impractical for the user to keep track of all context changes and issue explicit discovery requests. Instead, implicit service discovery is crucial, i.e. after each change in the user context, the client should be updated with the set of the currently most interesting services.

In order to be able to deal with all 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 them with the user preferences. A smart home system based on on this formal model and the discovery algorithm needs to be able to automatically detect changes in the user's context, initiate a new service discovery and update the user's client with the discovery result.

3 Hyperspace analogue to context

Hyperspace Analogue to Context (*HAC*) is a formal model of context as a multi-dimensional space, effectively capturing continuously changing status from various context sources. HAC is a novel approach to modeling context. It is intended to avoid the symbol grounding problem of ontology-based models [9]. The relationships between context properties in HAC are characterized by geometric structures, which largely improves the performance when reasoning on situations. 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 only use a fraction of all dimensions, as will be shown later.

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 $< d_{\text{location}} = bedroom, d_{\text{physical}} = normal >$. When an emergency happens, Frida is moved to another point: $< d_{\text{location}} = bedroom, d_{\text{physical}} = abnormal >$.

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. $< D_{\text{humid}} = [60 \ldots 70], D_{\text{temp}} = [15 \ldots 18] >$ 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 $b_i = 0 \iff D_i = \emptyset$. The basis of a context

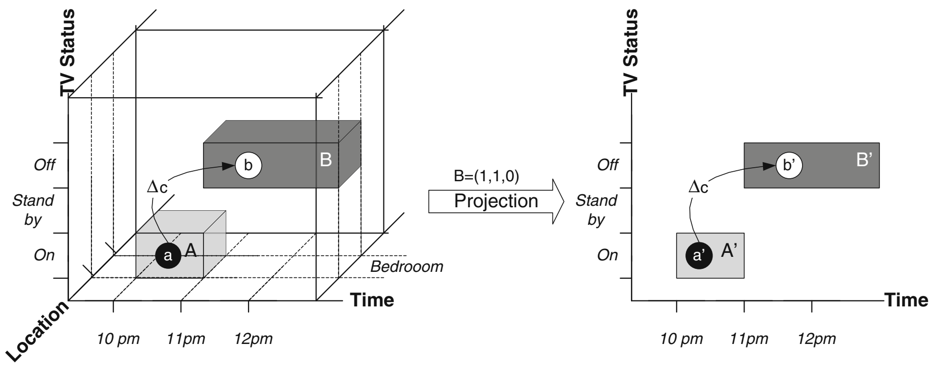

Figure 1 Visualization of concepts.

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 (Projection) Projection is an operation to render a partial view of context. It is denoted by $\times B$. $C' = C \times B$, where $\forall i, (D'_i = D_i \iff b_i = 1) \wedge (D'_i = \emptyset \iff b_i = 0)$.

This operation can be applied to both context scopes and context points. 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 the projection $C \times B(c) =< \emptyset, D_2, \emptyset >$.

Definition 6 (Context change) Context change $\times \Delta c =< \Delta d_1, \Delta d_2 ... \Delta d_n >$, is an operation to change a context point. Δd_i denotes the new value for a dimension, $d'_i = d_i \times \Delta d_i$. If d_i doesn't change, $\Delta d_i = \emptyset$.

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.

Figure 1 visualizes the concepts introduced in this section. The example context space has three dimensions, $\mathbb{H} =< \mathbb{D}_{\text{Time}}, \mathbb{D}_{\text{TVStatus}}, \mathbb{D}_{\text{Location}} >$. Two context scopes A and B are defined in the space. In each scope a context point, a and b respectively, is defined. Assume that the user has the current status a. If now the time changes from 10:30 to 11:10 and the TV is turned off, then the user moves to b; $\Delta c =< 11{:}10, off, \emptyset >$. The context change can be characterized on a projected 2-dimensional space with the projection operation using a basis $B =< 1, 1, 0 >$.

3.3 Services and user preferences

Definition 7 (Context-aware Service) A context-aware service is situated in HAC. It can be invoked in a certain context scope and invoking it will change the context of the user. Thus, a service $s =< C_s^{\text{Pre}}, C_s^{\text{Eff}} >$ is characterized by two types of context scopes.

- Precondition C_s^{Pre} is the required triggering condition of service s. When user is in C_s^{Pre}, service s becomes one possible choice. Formally, if c^u is the current context of the user, then $c^u \times B(C_s^{\text{Pre}}) \in C_s^{\text{Pre}}$.
- Effect C_s^{Eff} is the possible context scope after running a service. Formally $c^u \times B(C_s^{\text{Eff}}) \in C_s^{\text{Eff}}$.

The Precondition and Effect of context-aware services is similar to the Precondition and Effect in the IOPE (Input, Output, Precondition and Effect) model defined by M. Paolucci et al. [20]. If a service s is successfully executed, a transition of user context $c^u \rightarrow c'^u$ happens such that $c^u \times B(C_s^{\text{Pre}}) \in C_s^{\text{Pre}}$ and $c'^u \times B(C_s^{\text{Eff}}) \in C_s^{\text{Eff}}$. An example service s which turns on the oven and heats it up to a desired temperature could be described as $C_s^{\text{Pre}} =< D_{\text{oven}} = [off] >$ and $C_s^{\text{Eff}} =< D_{\text{oven}} = [on], D_{\text{ovenTemp}} = [120 \ldots 250] >$. It can be seen that most descriptions of preconditions and effects only use a fraction of all available dimensions, e.g. the oven service can be described by using only two of the potentially hundreds of dimensions.

According to Definition 4, the basis of the precondition and effect of each service is a n-dimensional vector, indicating which dimensions are relevant to the service. As we will show later, 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. $\mathbb{P}^u = \{(w_1, P_1), (w_2, P_2), \ldots, (w_t, P_t)\}$, 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 that matches a preference 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. Again, only a fraction of all possible dimensions will typically be used for describing a user preference. 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, 26] 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.

3.4.1 Location

The location $\mathbb{D}_{\text{location}}$ 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 the user. e.g. *in the kitchen* or *in front of the house*.

3.4.2 Time

The time $\mathbb{D}_{\text{time}}$ 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 machine is finished)*.

3.4.3 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 $\mathbb{D}_{\text{temperature}}$, the humidity $\mathbb{D}_{\text{humidity}}$ or the noise level $\mathbb{D}_{\text{volume}}$.

3.4.4 Health status

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

3.4.5 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 $\mathbb{D}_{\text{mediaStatus}}$ *(on, off, play, pause, forward)* or the coffee machine $\mathbb{D}_{\text{coffeeStatus}}$ *(on, off, making coffee)*. Quality of Service (QoS) [7, 21] 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 effect 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 [20]. In context matching we are not only looking for full matches, i.e. a service 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 is covered by the service's effect.

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 (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 $dmatch(D^1, D^2) = 0$.

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

The overall matching score is determined according to (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.

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

4.1.2 Numerical encoding

The actual calculation of the dimension overlap depends of course on the data type 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 [10] than with numerical values. Figure 2 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.

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

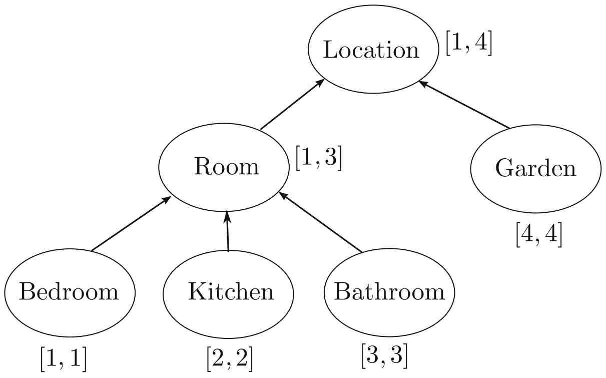

Figure 2 Taxonomy of location concepts.

postorder number among the descendants of the nodes. Figure 2 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 ontology coding scheme and matching method, this section presents our proactive algorithm for service discovery in HAC. The efficiency of the algorithm is assured by four 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 that are not affected by the changed context in the result set. Third, we introduce a separate offline algorithm for maintaining and evaluating service scores. Fourth, we describe a change detection algorithm that decides, which changes in the smart home environment should trigger a new service discovery.

Figure 3 illustrates the relationship between the algorithms presented in this section. The input parameters of the algorithms 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, $\emptyset$. 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's preconditions and effects are represented as numeric values according to the concept coding mechanisms presented in Section 4.1.

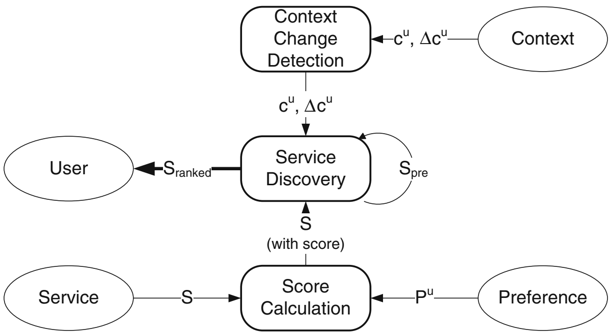

Figure 3 Illustration of algorithm input and output.

- 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 to evaluate which dimensions have changed.
- $\mathbb{P}^u$ is the set of user preference as defined in Definition 8.

Algorithm 1 Proactive Service Discovery

```
1: procedure ServiceDiscovery($S_{\text{pre}}, S, c^u, \Delta c^u$)
2: $S_{\text{cand}} = \emptyset$
3: for $\forall s \in S$ do
4:     if $B(\Delta c^u) \wedge B(C_s^{\text{Pre}}) \neq 0$ then
5:         $S_{\text{cand}} = S_{\text{cand}} + s$
6:     end if
7: end for
8: $S_{\text{ranked}} = S_{\text{pre}} - S_{\text{cand}}$
9: for $\forall s \in S_{\text{cand}}$ do
10:     $c'^u = c^u \times B(C_s^{\text{Pre}})$
11:     if $c'^u \in C_s^{\text{Pre}}$ then
12:         $S_{\text{ranked}}.add(s, s.score)$
13:     end if
14: end for
15: return $S_{\text{ranked}}$
16: end procedure
```

Algorithm 1 runs continuously in response to the update of context information. We call each execution of the algorithm a round.

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's precondition is evaluated: $B(\Delta c^u) \wedge B(C_s^{\text{Pre}}) \neq 0$. 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. A service is not affected by a context change, if none of the dimensions that are used in the service's precondition changed. This service is therefore also available in the new context and it is unnecessary to re-evaluate it. 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 in lines 9–14. Line 10 keeps in c'^{u} only the dimensions related to the precondition of a service. If c'^{u} is in the scope of service precondition C_s^{Pre} (line 11), this service will be included in the algorithm result. The service is added to S_{ranked} ordered by the services' scores. The service score is depending on the user preferences. We assume that user preferences are changing slowly and that the service scores are mostly fixed. We therefore calculate the service score not during the discovery phase, but in an extra algorithm called only when service scores have to be re-calculated.

4.3 Calculating service scores

Algorithm 2 evaluates how well services can cover the given user preferences. Again, only those dimensions of the service effect that are relevant to the user preference are considered (line 5). For each of the preferences, the matching method described in Section 4.1 is invoked in line 6. 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 7–8). In our system, the algorithm is at first run at initialization phase. When the system is running, this algorithm is run in background only for preference changes and updates in the list of available services.

Algorithm 2 Service Score Calculation

```
1:  procedure ServiceScore($S$, $\mathbb{P}^u$)
2:      for $\forall s \in S$ do
3:          $s.score = 0$
4:          for $\forall (w_i, P_i) \in \mathbb{P}^u$ do
5:              $C_s'^{\text{Eff}} = C_s^{\text{Eff}} \times B(P^u)$
6:              $score = w_i * Matching(P_i, C_s'^{\text{Eff}})$
7:              if $score > s.score$ then
8:                  s.score=score
9:              end if
10:         end for
11:     end for
12: end procedure
```

4.4 Detecting significant context changes

We would like to avoid running a new round of the service discovery algorithm for each small context update that occurs in the house. A temperature sensor may, for example, send frequent context updates with the individual temperature readings differing only by 0.1°C. Instead of re-running the service discovery for each of those updates, we need to identify the significant changes in the context. What is considered a significant context change differs of course between the dimensions. We therefore define for every dimension in HAC a sampling function $samp(d)$, which converts the possible context values into a finite, discrete value set. We consider a context change from d_i to d'_i for a dimension $\mathbb{D}_i$ to be significant, if $samp_i(d_i) \neq samp_i(d'_i)$. In case of the temperature dimension, for example, the sampling function could round the temperature reading to the largest previous integer. Say that the temperature at the beginning is 16.1°C. Continuous increases of 0.1°C result only in a significant context change, if the temperature raises to at least 17.0°C. At this point the sampled value jumps from 16°C to 17°C and a service discovery is triggered.

We see two major advantages of using a sampling function instead of a simple threshold for detecting significant context changes. First, using a sampling function makes it easy to keep track of continuous, small updates of a context value. Second, a sampling function allows defining a different behavior for different subsets of a dimension. For example for a dimension representing the room temperature, the sampling function could generate a much more fine-grained discretization for temperature values between 18 and 23°C than for values between 23 and 30°C.

The Change Detection component runs Algorithm 3 whenever context updates arise. The input to the algorithm is the user's current context $c^u =< d_1, d_2, ..., d_n >$ and a context point Δc that describes the updated context information. Line 2 applies the context update to the user's current context to obtain the new user context $c'^u =< d'_1, d'_2, ..., d'_n >$. Lines 3–7 check for all updated dimensions, if the context update was significant. Line 4 checks, if the sampled value of the old user context equals the sampled value of the updated context. If this is the case, then line 5 sets the basis of this dimension to zero in Δc, marking the dimension thereby as unchanged. If for at least one dimension the context update was significant, the service discovery is initiated (lines 8–10).

Algorithm 3 Detecting significant context changes

```
 1: procedure ContextChangeDetection($c^u$, $\Delta c$)
 2:     $c'^u = c^u \times \Delta c$
 3:     for $\forall d'_i \in \Delta c'$ do
 4:         if $samp_i(d'_i) = samp_i(d_i)$ then
 5:             $B(d'_i) = 0$
 6:         end if
 7:     end for
 8:     if $B(c') \neq 0$ then
 9:         Invoke Algorithm 1
10:     end if
11: end procedure
```

5 A HAC-based smart home system

5.1 System architecture

We have implemented a prototype of a pervasive system that incorporates our formal model of context. The prototype is based on the scenario of a smart home

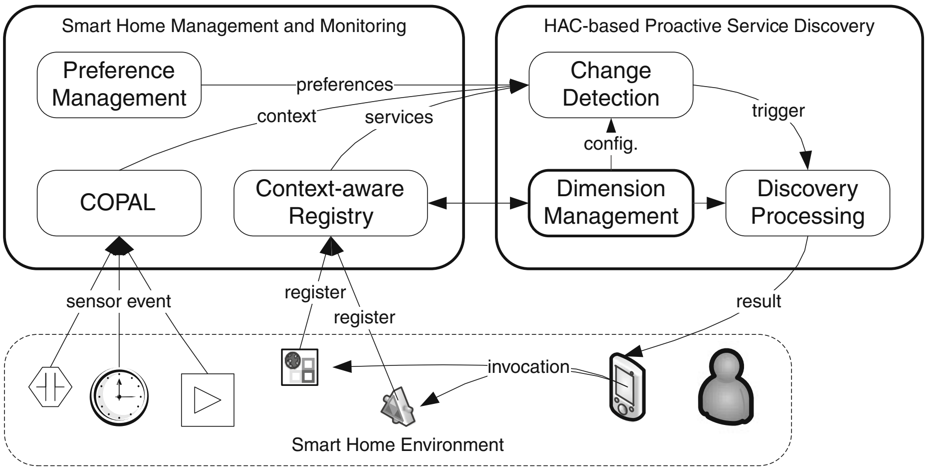

Figure 4 System architecture.

system. In the following we will first give an overview of the system architecture and will afterwards take a closer look at our system by detailing two typical situations: *(i)* installing new context and service providers and *(ii)* reacting to changes in the environment.

Figure 4 shows the architecture of our system. Devices and sensors are deployed throughout the smart home environment, providing context information and services that can be invoked by the user. The *Context-aware Registry* administers information about services that are deployed in the system. Whenever a new device gets installed, it must register itself with the Context-aware Registry, which will create and register a context-aware description. Since HAC is a general concept and specific pervasive environments can have different sets of context dimensions depending on the available devices and user requirements, the *Dimension Management* component is introduced to customize the types of context information. It provides information about all dimensions currently defined in the system to the other components. User preferences are managed in the *Preference Management* component.

Context information from heterogeneous sources is collected and processed by the COPAL (COntext Provisioning for ALl) framework,[1] which is a complex event processing system, producing formatted context events. COPAL features a composable architecture allowing to integrate new context sources, to create new information models, and to support various information processing requirements of context-aware services.

The *Change Detection* component monitors significant changes of context, user preferences and services. On detection of such a change, it triggers the service discovery in the *Discovery Processing* component which is running our discovery algorithm. The Discovery Processing automatically updates the user's client with the discovery results.

[1] http://www.infosys.tuwien.ac.at/m2projects/sm4all/copal/

5.2 Adding context and service providers

One of the foundations of smart home environments is the rich selection of context information providers that are deployed throughout the house. The majority of context information is provided by sensors which for example monitor environmental properties, the user's health status or detect hazardous situations. When a new context provider is deployed in the house, the context information that can be provided has to be described. A temperature sensor may, for example, describe a context type for the temperature containing numerical data between −10 and 70°C. Sources of context information are registered to COPAL as context publishers.

As described in Definition 7, the capabilities of context-aware services are based on the available context information. Service providers are for example house controllers, household appliances or entertainment devices. For deploying a new service provider, all services offered need to be described with their preconditions and effects, referring to the context types available in the system. An air-conditioner may be described with preconditions and effects referring to the context type defined by the temperature sensor. Service descriptions include of course also the name of the service, the URI where the service can be called and information about service parameters.

The devices deployed in a smart home are often both context and service providers. A lamp for example offers context information about whether it is currently turned on or off. It also offers services for turning it on and off. The service's preconditions and effects are referring to the lamp status, e.g. the service s for turning the lamp on may be defined with precondition $C_s^{\text{pre}} =< lampStatus = off >$ and effect $C_s^{\text{pre}} =< lampStatus = on >$.

Figure 5 shows the sequence of action happening when a new context or service provider is being registered. A description of the new device is sent to the Context-aware Registry. For each new context type defined in the device description, the registry creates a new context dimension with a globally unique name and registers it with the Dimension Manager. The Dimension Manager makes the information about a new context type known to the Context Provisioning framework.

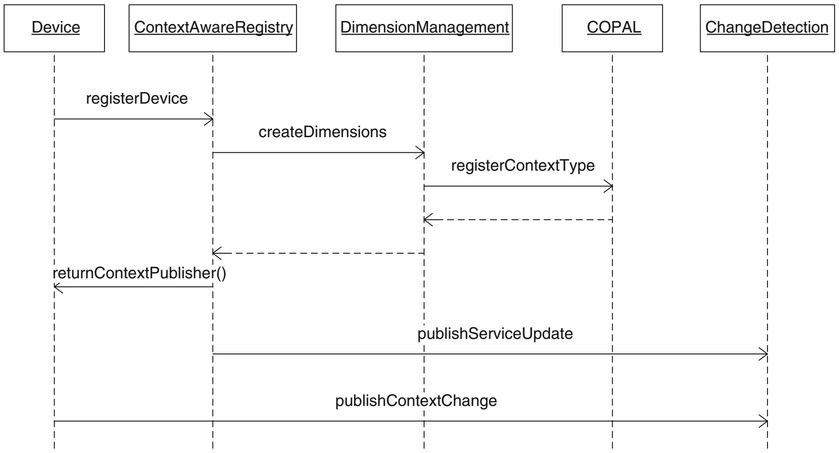

Figure 5 Adding a new context-aware device.

```
@Path("coffeeMachine")
@ContextAwareDevice("myCoffeeMachine")
@ContextTypes({@ContextType="status",
  values = "http://www.sm4all-project.eu/types.owl#CoffeeMachineStatus"})
public interface CoffeeMachine
{
    @POST
    @Path("coffee")
    @ContextAwareService(name="Make␣coffee",icon="coffeeCup.jpg")
    @Precondition({@ContextElement(type="status",value="idle")},
                  {@ContextElement(type="Location" ,value="Kitchen")})
    @Effect({@ContextElement(type="status",value="in_use")})
    void makeCoffee(String coffeeType);

    ...
}
```

Figure 6 Annotations describing a context-aware coffee machine.

For all offered services, the Context-aware Registry creates the HAC-based description of the preconditions and effects. It registers this information and also publishes information about the new services. After the registration process is finished, the Context-aware Registry returns a reference to a COPAL Context Publisher for publishing updates of the context values. The device will typically publish initial values of the context information it provides.

We propose a simple way of describing service and context providers using Java annotations. Figure 6 shows the description of a coffee machine using these annotations. The coffee machine is marked as a context-aware device with the name *myCoffeeMachine*. A new context type *status* is defined. The definition refers to an OWL ontology that contains the allowed values for the type (e.g. idle and in use) and will be the basis for the encoding of this new context type. A service is defined for making a cup of coffee. The service's precondition refers to the new context type *status*, declaring that the service should only be available when the coffee machine is currently idle. It also refers to the context type *Location*, stating that the service is available when the user is currently in the kitchen. References to context types already defined, e.g. *Location*, are automatically resolved during device deployment. The effect of the service is, that the coffee machine is marked as in use.

When the coffee machine is deployed to the system, the annotations are automatically read and a valid device description is created and used for setting it up in the system. As it can be seen in Figure 6, it is possible to mix our device annotations with annotations for RESTful web services [8]. In this case our system reads both the device and the REST annotations, deploys the web service to a service container and automatically sets up all necessary information for service invocation, i.e. service URI, call method and service parameters. Enabling the communication with the actual physical devices is out of the scope of this paper, but is under investigation in the SM4All project.[2] In the project a tool set is being developed, that aims at making the functionalities of smart home devices using standard technologies like UPnP and Bluetooth easily available through web service interfaces.

[2] http://www.sm4all-project.eu

Table 1 Discovery actions for changes in the smart home environment.

Detected change	Action
Significant context change Δc^u	$ServiceDiscovery(S_{\text{ranked}}, S, c^u, \Delta c^u)$
New service s	$ServiceScore(s, \mathbb{P}^u)$ $if\ (c^u \times B(C_s^{\text{pre}})) \in C_s^{\text{pre}}$ $S_{\text{ranked}}.InsertOrdered(s)$
Removed service s	$S_{\text{ranked}} = S_{\text{ranked}} \setminus s$
New preference P	$ServiceScore(S, P)$ $S_{\text{ranked}}.updateOrdering(S)$
Removed preference P	$ServiceScore(S, \mathbb{P}^u \setminus P)$ $S_{\text{ranked}}.updateOrdering(S)$

5.3 Reacting to changes in the smart home environment

Changes in the user's current context, the set of registered services or the set of user preferences may result in the need to update the current list of discovered services. The Change Detection component listens to any such changes and detects the significant changes. If necessary it triggers a new service discovery with the Discovery Processing component. Based on the type of the detected change, the Discovery Processing executes specific actions to update the list of discovered services. Table 1 gives an overview of the actions that are executed. If a significant context change was detected, the service discovery process is executed as described in Section 4. If a new service was registered, first the score of this service is calculated. If the service is applicable in the current context, it is added to the set of ranked services. If a service was unregistered, it is removed from the set of ranked services.

For changes in the set of preferences, the score of all services has to be recalculated. Since the set of services discovered in S_{ranked} is not affected by the change of preferences, it is enough to reorder the ranked services based on the new service scores. Any changes in the ranked services result in a service update that is sent to the user client, which then updates it's display of services accordingly.

5.4 Android client

For testing our prototype of a HAC-based smart home system, we have implemented a client for Android-based smart phones. A screenshot of the client can be seen in Figure 7. The major part of the display is dedicated to the list of currently available services. The client needs to be connected to the Discovery Processing to receive an initial service list and subsequent updates of the currently available services. A status icon at the top of the client interface shows the clients connection status and allows to connect to and disconnect from the Discovery Processing using a socket connection. When the client connects to the Discovery Processing, it must identify itself with the user's credentials and receives an initial service list calculated according to the user's current context and preferences. Subsequent updates are sent to the client whenever the set of currently available services changes.

The service update contains the identificators of the discovered services ordered by the service scores. If the client receives a new service list, it needs to resolve further information about the services. It gathers from the Context-Aware Registry information about the service names, icons, parameters and where and how to call

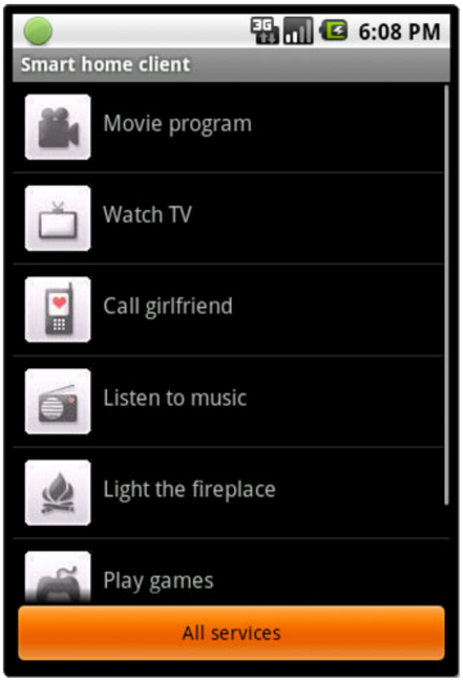

Figure 7 Screenshot of the Android client.

the services. To avoid having to retrieve service information for every update of the service list, the client keeps a cache of the service details in a local database.

Services can be invoked by clicking on them in the user interface; the client then displays a notification about success or failure of the service invocation. Invoking a service may of course mean that the current context changes and that the service list will be updated again shortly.

6 Experiments

The performance of the service discovery significantly influences the user's experience. Context changes can happen very frequently and the discovery results should quickly reflect these changes. In this chapter we first evaluate the performance of the discovery algorithms and the relevance of the discovered services using synthetical data. Afterwards we evaluate the prototype of our context-aware pervasive system based on our scenarios of a smart home.

6.1 Performance evaluation

In order to test how well our formal model of context and our discovery algorithm can handle frequent context changes, we have run a series of experiments using synthetically generated data. We have identified three important variables that can influence the performance of the discovery algorithm: *(i)* the number of available services, *(ii)* the number of dimensions, *(iii)* the percentage of services affected by a context change (in the following referred to as affected services).

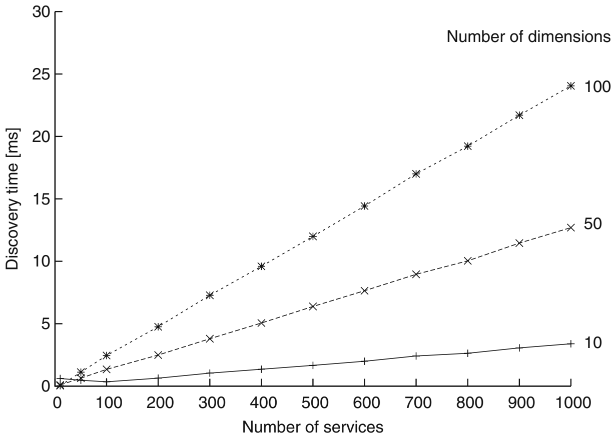

Figure 8 Discovery time depending on number of services and dimensions (30% affected services).

We have performed several experiments to show the system performance in different settings. For each of the experiments we have generated synthetic data 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 affected—in a real home with a high service diversity we expect a much smaller number of affected services, which will in turn result in a 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 8 shows how the discovery time depends on the number of services and dimensions. It can be seen that the system scales linearly for the number of dimensions and services. With 1000 services and 100 dimensions, the discovery time equals to circa 25 ms (with a standard deviation of 2.2 ms), allowing for a near real-time update of the user interface.

Figure 9 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

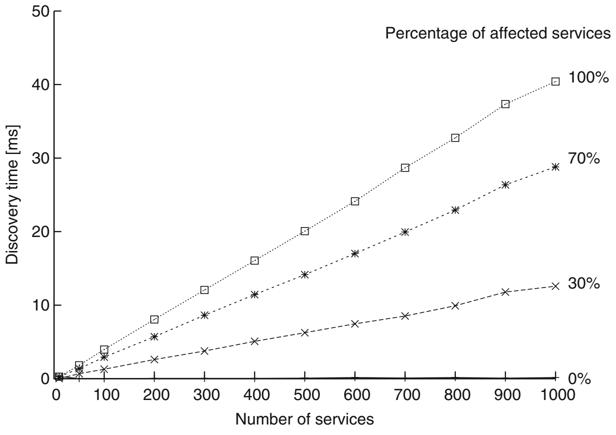

Figure 9 Discovery time depending on number of services and percentage of affected services (50 dimensions).

case only the bit-set operations to identify the services affected by the context change need to be performed. For a higher percentage of affected services, the time needed to check which services can be used in the current context increases. We can see that the system also scales well in this regard, for 100% affected services, the discovery time equals circa 40 ms (with a standard deviation of 1.1 ms).

We have also evaluated Algorithm 2 for Score Calculation to find out how fast the service scores can be calculated depending on the number of services and preferences. For the initial calculation of service scores, all combinations of services and preferences have to be evaluated. We have found that for 100 services and 10 preferences with 50 dimensions the calculation time is circa 35 ms, for 1000 services and 50 preferences it is circa 350 ms.

As described in Section 5.3, it is not necessary during runtime to recalculate all combinations of services and preferences. When new preferences are added, it is sufficient to calculate the scores for the new preferences and all services; analogously the scores only have to be calculated for the new services when adding services. Figure 10 shows the performance of the score calculation algorithm when adding ten new preferences depending on the number of services currently registered. It can be seen that the calculation algorithm scales linearly; when increasing the number of services from 100 to 1000, the calculation time increases by a tenfold, equaling circa 75 ms (1.2 milliseconds standard deviation).

6.2 Relevance of discovery results

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 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 (3), in which t is the number of

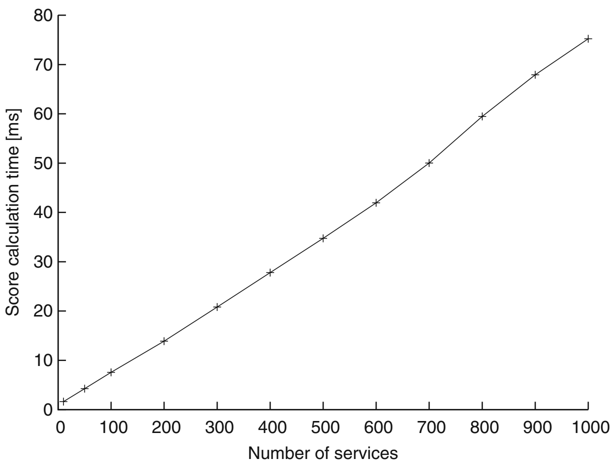

Figure 10 Score calculation time for 10 new preferences (50 dimensions).

preferences. Similarly the output context of all services is merged to *ServAll* and that of the k selected services to *ServSel*. We then calculated how well the preferences are covered using (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 11 shows the results of this experiment for a varying number of icons. Even with only 9 service 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 limitations of the user interface and the number of interesting services.

6.3 Prototype evaluation

We have evaluated how fast our prototype of a HAC-based smart home system can react to changes in the environment. We have therefore measured the round-trip time beginning with the user executing a context-changing action and ending with the user receiving an updated service list. This round-trip time includes: *(i)* invocating a service on the user device, *(ii)* the service executing and issuing a context change, *(iii)* the Change Detection reacting to the context change and triggering a service discovery, *(iv)* the Discovery Processing executing the discovery and sending the results to the client and *(v)* the client updating its service display.

In our evaluation system we have deployed all components of our smart home system (Context-Aware Registry, Change Detection, Discovery Management, Dimension Management, Preference Management and Context Provisioning) on the same host. This is a realistic set-up of a smart home system with a central

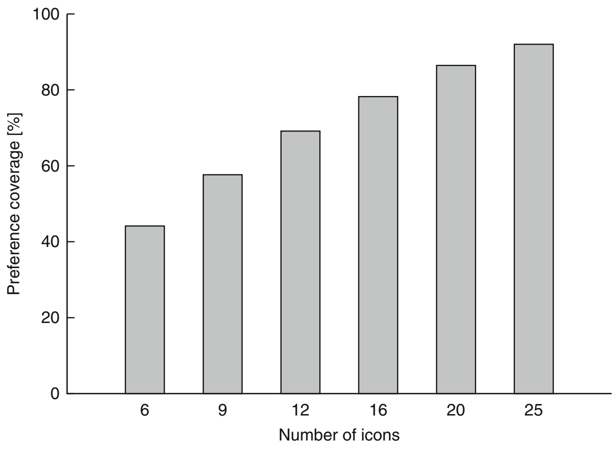

Figure 11 Preference coverage depending on the number of icons displayed.

home management server. The user client is deployed in the same wireless LAN network as the central server. Based on our two usage scenarios, we have created a set of circa 30 context-aware devices implementing all necessary devices, sensors and human actors. Currently we are simulating all devices. Since our evaluation focuses at measuring how fast our smart home system can react to any changes in the smart home environment, we assume that all context changes happen instantly. For example, when calling the *OpenBlinds* service, the context changes instantly to $\mathbb{D}_{\text{blinds}} = open$; whereas for actual blinds this status would be reached only after a certain time.

We have first tested the performance of our system using a simple command-line client. We have measured the round-trip time for 100 replications and found that the new service list is received by the client circa 25 ms after issuing a service invocation (11 ms standard deviation). Using the same setup we have then tested our Android client. The round-trip time here starts when the user clicks on a service and finishes when the new service list is displayed. We have found that the round-trip time in this case averages to circa 150 ms. The increased average compared to the command-line client can be attributed to the necessary reload of the user interface.

Our experimental results show, that our smart home system can provide clients with an up-to-date service list in real time. Indeed, for our smart phone client, the time needed for processing a change in the smart home environment and the subsequent service discovery is minimal compared to the time needed for managing the graphical user interface.

7 Related work

Service discovery in pervasive environments has been intensively investigated in recent years [24, 25]. 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.*

The following two examples address service discovery with QoS and context information, but they don't consider the impact of continuous context changes on service discovery. Mokhtar et al. [18] proposed the EASY (Efficient semantic Service discoverY) framework which takes QoS and context into account. EASY relies on semantic modeling using OWL-S to describe context, QoS and functionalities of services. The EASY-L language is proposed to present service capabilities and requests; the corresponding matching approach is called EASY-M. For improving the performance of service discovery, the semantic description of services is optimized by a numeric coding scheme, a widely adopted method for enhancing the performance of ontology processing. User preference is not considered in EASY. Our focus in information modeling is not on service descriptions but on modeling powerful context descriptions that allow us to provide context-driven discovery results without explicit service requests. Different numeric coding schemes for basis and context concepts are an essential aspect of our context model. Our prototype includes a specialized context provisioning framework so that the updated context information can be dealt with in real time. Park et al. [21] 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. The distance between users and services is calculated based on the virtual space parameters. The idea of a VPS is similar to HAC, however we extensively elaborated on the modeling of virtual spaces, dimensions and the calculation of similarity. Again the factors considered in VPS are service-oriented rather than context-oriented.

The *Context attributes* proposed by Lee and Helal [13] are an early effort of service discovery in smart home environments. Context attributes extend the Jini [3] service discovery protocol to include domain specific context information in the service description. The attributes are dynamically evaluated when discovering services. The processing of attributes is transparent to the client, but the evaluation rules are hard coded in source code, which largely limits the flexibility of the approach. Similarly, Cuddy et al. [7] have applied generic dynamic attributes to service descriptions. The corresponding evaluation approach relies on application-specific weight vectors to tune the significance of each dynamic attribute. Broens et al. [6] enhanced the idea of context attributes with an ontology-based context model. The model is able to describe the relationships between different context attributes, and in turn support the semantic matching of service and context description. All of these works focus more on an extension of existing service discovery protocols rather than on the dynamic and complex nature of context.

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 et al. [5] proposed a user-centric service view for explicit discovery. Discovery scope and service view are the keys to indicate the changes of context information and its impact on available services. The paper illustrates the implementation of the two concepts, but it is unclear how the discovery scope and service view are modeled and which factors were driving their definition. Hesselman et al. [12] presented the idea of a persistent discovery request but did not provide further details. Our goal is similar to these two in terms of updating the set of relevant services dynamically. However our work is based on a comprehensive formal model of context and an efficient algorithm able to reflect the changes in context, user preference, and services in real time. Our prototype is implemented with a strong focus on maintaining context information and detecting changes, which we regard as keys to effectively changing discovery scopes. The numerical coding approach and a series of discovery scope optimizations improve the performance of the iterative service discovery significantly. As a result, the discovery scopes can be more fine-grained and more frequent proactive discovery can be performed to improve user experience.

Our HAC has been inspired by (HAL) Hyperspace Analogue to Language [17] and the context space of assumptions [14]. 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 interpretations of assumptions within different communication contexts, which is completely different to the meaning of context in pervasive environments. In our domain, Padowitz et al. [19] 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, since based on HAC, we model services and user preferences to propose a context-driven service discovery approach.

8 Conclusion

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 interesting 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. Existing service discovery results are efficiently reused to minimize the need for context matching. The calculation of service score against user preferences is carried out in a separate algorithm, that is only invoked when changes in services or preferences happen. An efficient sampling method is used to detect significant context changes and subsequently triggers service discovery.

We have proposed and implemented a context-driven proactive service discovery system based on HAC. A context-provisioning system (COPAL) is employed to dynamically update context information and manage context sources. New services and new context types can be added to a context-aware registry by an annotation-based service description approach. A client software based on the Android smart phone platform was developed to present proactive discovery results to the user. Our experimental results prove that our system can efficiently and effectively provide users with up-to-date information about the most relevant and interesting services. In addition, our context model and discovery approach could also assist traditional, explicit service discovery approaches by limiting their search scope.

References

1. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: Proceedings of the First International Symposium on Handheld and Ubiquitous Computing, pp. 304–307 (1999)
2. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. In: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, vol. 18, pp. 253–262. ACM, New York (1989)
3. Arnold, K., Scheifler, R., Waldo, J., O'Sullivan, B., Wollrath, A.: Jini Specification. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
4. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. Int. J. Ad Hoc Ubiquitous Comput. **2**, 263–277 (2007)

5. Bellavista, P., Corradi, A., Montanari, R., Toninelli, A.: Context-aware semantic discovery for next generation mobile systems. IEEE Commun. Mag. **44**(9), 62–71 (2006)
6. Broens, T., Pokraev, S., van Sinderen, M., Koolwaaij, J., Dockhorn Costa, P.: Context-aware, ontology-based service discovery. In: Ambient Intelligence. Lecture Notes in Computer Science, vol. 3295, pp. 72–83. Springer, Berlin/Heidelberg (2004)
7. Cuddy, S., Katchabaw, M., Lutfiyya, H.: Context-aware service selection based on dynamic and static service attributes. In: IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, 2005, (WiMob'2005), vol. 4, pp. 13–20 (2005)
8. Fielding, R.T., Software, D., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Internet Technol. **2**, 115–150 (2002)
9. Gärdenfors, P.: Conceptual Spaces: the Geometry of Thought. MIT Press (2004)
10. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993)
11. Guger, C., Holzner, C., Groenegress, C., Edlinger, G., Slater, M.: Brain-computer interface for virtual reality control. In: Proceedings of ESANN 2009, pp. 443–448 (2009)
12. Hesselman, C., Tokmakoff, A., Pawar, P., Iacob, S., et al.: Discovery and composition of services for context-aware systems. Lect. Notes Comput. Sci. **4272**, 67 (2006)
13. Lee, C., Helal, S.: Context attributes: an approach to enable context-awareness for service discovery. In: Proceedings of the 2003 Symposium on Applications and the Internet, pp. 22–30 (2003)
14. Lenat, D.: The dimensions of context space. Technical report, CYCorp. http://www.cyc.com/doc/context-space.pdf (1998). Accessed 2 July 2010
15. Li, F., Rasch, K., Truong, H.-L., Ayani, R., Dustdar, S.: Proactive service discovery in pervasive environments. In: Proceedings of the 7th ACM International Conference on Pervasive Services (ICPS), pp. 126–133 (2010)
16. Loke, S.W., Krishnaswamy, S., Naing, T.T.: Service domains for ambient services: concept and experimentation. Mobile Netw. Appl. **10**, 395–404 (2005)
17. Lund, K., Burgess, C.: Producing high-dimensional semantic spaces from lexical co-occurrence. Behav. Res. Methods Instrum. Comput. **28**(2), 203–208 (1996)
18. Mokhtar, S.B., Preuveneers, D., Georgantas, N., Issarny, V., Berbers, Y.: Easy: efficient semantic service discovery in pervasive computing environments with qos and context support. J. Syst. Softw. **81**(5), 785–808 (2008)
19. Padovitz, A., Loke, S., Zaslavsky, A.: Towards a theory of context spaces. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, pp. 38–42 (2004)
20. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: The Semantic Web ISWC 2002, vol. 2342, pp. 333–347. Springer, Berlin/Heidelberg (2002)
21. Park, K., Yoon, U., Kim, S.: Personalized service discovery in ubiquitous computing environments. IEEE Pervasive Computing **8**(1), 58–65 (2009)
22. Park, M., Hong, J., Cho, S.: Location-based recommendation system using bayesian users preference model in mobile devices. In: Ubiquitous Intelligence and Computing, pp. 1130–1139 (2007)
23. Truong, H.L., Dustdar, S.: A survey on context-aware web service systems. Int. J. Web Inf. Syst. **5**(1), 5–31 (2009)
24. Ververidis, C., Polyzos, G.: Service discovery for mobile ad hoc networks: a survey of issues and techniques. IEEE Communications Surveys & Tutorials **10**(3), 30–45 (2008)
25. Zhu, F., Mutka, M., Ni, L.: Service discovery in pervasive computing environments. IEEE Pervasive Computing **4**(4), 81–90 (2005)
26. Zimmermann, A., Lorenz, A., Oppermann, R.: An operational definition of context. In: Modeling and Using Context, pp. 558–571 (2007)