Discovery Pervasive Services based on their Expected Use

R. Kazhamiakin, V. Kerhet, M. Paolucci, M. Pistore, and M. Wagner

[1]FBK-Irst, via Sommarive 18, Povo, Italy
{raman,pistore}@fbk.eu
[2] Faculty of Computer Science, Free University of Bozen-Bolzano
kerhet@inf.unibz.it
[3] DOCOMO Euro-Labs, Landsbergerstrasse 312, Munich, Germany
{paolucci,wagner}@docomolab-euro.com

Abstract. Pervasive services accomplish tasks that are related with common tasks in the life of the user such as paying for parking or buying a bus ticket. These services are often closely related to a specific location and to the situation of the user; and they are not characterized by a strong notion of goal that must be achieved as part of a much broader plan, but they are used to address the contingent situation. For these reasons, these services challenge the usual vision of service discovery and composition as goal directed activities. In this paper we propose a new way to look at service discovery that is centered around the activities of the user and the information that the user has available rather than the goals that a given service achieves.

Keywords: Mobile&Telco Services, Pervasive Services, Service Discovery

1 Introduction

The last few years have seen the deployment of an increasing number of pervasive services that support everyday tasks of our life. Such services range from SMS-based services for paying for parking in a number of European cities such as Milan[1] and Vienna to train ticketing in Germany[2], to RFID-based food selection at McDonalds' restaurants in Seoul[3].

Pervasive services, like the ones listed above, provide two major challenges to service oriented computing. First, they are closely related to specific locations and context of the user, because they are provided through short range protocols such as RFID and Bluetooth, or because their use is intrinsically related to the environment in which the user operates, as in the case of the SMS-based parking services. Second, these services challenge the idea that service discovery and composition are goal oriented processes; rather users exploit these services without ever creating an explicit

[1] www.atm-mi.it/ATM/Muoversi/Parcheggi/SOSTA_MILANO_SMS.html (l.v. 15.12.08)
[2] http://www.bahn.de/p/view/buchung/mobil/handy_ticket.shtml (l.v. 22.03.09)
[3] http://www.koreatimes.co.kr/www/news/tech/2007/09/133_10034.html (l.v. 15.12.08)

notion of goal. Consider for example a performance ticket: to go to the performance, the user may take advantage of a number of services such as *navigation services* to go to the show, *"Point of Interest" service* to suggest restaurants where to eat nearby the theatre; *on-line social services* to share comments and suggestions with friends; contextual services such as parking services which depend on the exact location of the parking: on the street side vs in a private parking lot. Crucially all these services are of interest for the user and provide added value to the performance ticket, and yet the performance ticket itself is neither the input of any of them, nor these services contribute to explicitly stated goals of the user.

Since there is no explicit notion of goal, discovery algorithms that are based on input/output matching (see [9,11] among the many others) or on precondition and effect matching [1], and possibly quality of service [12,13] hardly apply. Instead, there is a need of different algorithms that organize the available services around the tasks that the user is pursuing. To this extent, in this paper we propose *use-based service discovery* as a new framework for service discovery that aims at matching services with the user's tasks. The resulting discovery framework *recommends* services that may be useful in the given situation, leaving to the user the task of deciding whether to use them or not.

The intuition behind Use-based discovery is to rely on a trend that sees the mobile phone as a collector of very important pieces of information about our life. Already now objects like Deutsche Bahn's "handy ticket", a train ticket delivered to the mobile phone[4], and electronic boarding passes provided by airCanada[5] among the many other airlines are evidence of this trend. Crucially, these objects are evidence of (possibly future) activities of the user. The objective therefore is to envision a service discovery mechanism that given a new object, such as a new ticket, selects the known services that are relevant for this object; and, conversely, that given new services selects the user's objects to which they most likely apply.

The results of Use-based discovery is a contextualized discovery mechanism that reacts to changes in the life of the user trying to support her through her daily life by offering services that may address problems of the user. Ultimately, Use-based discovery is an essential piece to transform the mobile phone from a collector of the user's data to an organizer of such data.

The rest of the paper is organized as follows. First, we address the technological problems related with Use-based discovery, namely section 1 addresses the representation of services and section 3 the matching algorithms. We will then proceed with an evaluation discussed in section 4.1; and finally, in section 5 we will discuss the remaining open problems and conclude in section 6.

2 Specification of Service Advertisement

The first problem to address when facing a new discovery algorithm is the process of service advertisement and the description of the services. Mobile phones have many different ways discover available services. These range from UPnP protocols [3], to

[4] http://www.bahn.de/p/view/buchung/mobil/handy_ticket.shtml (l.v.4.4.2009)
[5] http://www.aircanada.com/en/travelinfo/traveller/mobile/mci.html (lv 24.06.2009)

Bluetooth discovery[9], to the reading of 2D bar codes such as QR Codes[10] or Near Field Communication transmission [8], to P2P interactions [14] to the simple typing of URLs of service advertisements. Whereas, these algorithms concentrate in finding the services, there is though very limited support into organizing the services so that they can be automatically associated to the activities of the user, and invoked.

The intuition behind Use-based discovery is that services, exploiting the channels reported above, advertise themselves to mobile phones by declaring for which type of objects a mobile user would ***use*** such service. More formally, we define advertisements through the relation $\underline{\texttt{USE}}\texttt{:S}\times\texttt{T}$ that maps services (`S`) to types (`T`), where types are classes in an ontology[11] that are used to specify for which types of objects a service may be used. For example, a navigation service may be described as to be "used" for locations, meaning that the developer of the service expects it to be good to handle location information.

Fig. 1 shows more in details the relation between types of data and services that are advertised. In the figure, the cyrcles represent data types organized in a hierarchy as they are known to a system. Specifically, on the right side the node marked with $\mathcal{M}$ corresponds to the class *Money*, the one marked with $\mathcal{T}$ that corresponds to the class *Time*, and the third with label $\mathcal{L}$ that corresponds to the class *Location*; on the left side there are three nodes labeled *Ticket*, *Train Ticket*, and *Flight Ticket* representing different types of tickets that are known to the system. In addition to the types, Fig. 1 shows the services that are known to be available. In the figure, services are represented as filled gray labeled hexagons; specifically in the figure are the following services are shown: `e-wallet`, `Calendar`, `Navigation`, `Track Info`, and `Check-in`. As the figure shows, services advertise themselves through the types to which they associate themselves. These types are decorated by empty hexagons overlapping on the corresponding types; as shown in the figure the `e-wallet` service is used in conjunction with data of type *Money*, the `Calendar` with data of type *Time*, the `Navigation` with data of type *Location*, the `Track Info` with data of type *Train Ticket*, and the `Check-in` with data of type *Flight Ticket* respectively.

Whereas in the figure services associate to fundamental classes, as pointed out above, service advertisements can exploit the whole expressive power of the ontology language, which is OWL in our case. Therefore, other restrictions can be expressed. For example, the navigation system may specify that it works only in specific locations such as the United States, or only in Europe, or only in a shopping mall by specifying that the relations `nav` $\underline{\texttt{USE}}$ *US*, `nav` $\underline{\texttt{USE}}$ *Europe* or `nav` $\underline{\texttt{USE}}$

[9] See www.**bluetooth**.org *(lv 25.06.2009)*

[10] See http://www.denso-wave.com/qrcode/index-e.html (lv 25.06.2009)

[11] Although our implementation of use-based discovery has been based on a description of the services in OWL and it relates the use to OWL classes, the discovery algorithms proposed do not necessarily depend on OWL, but rather on taxonomy of types which could also be modeled in Object Oriented programming. Using OWL ontologies still may take advantage of the richer expressivity of OWL compared with OO languages, as well as of the logic inference performed by OWL inference engines.

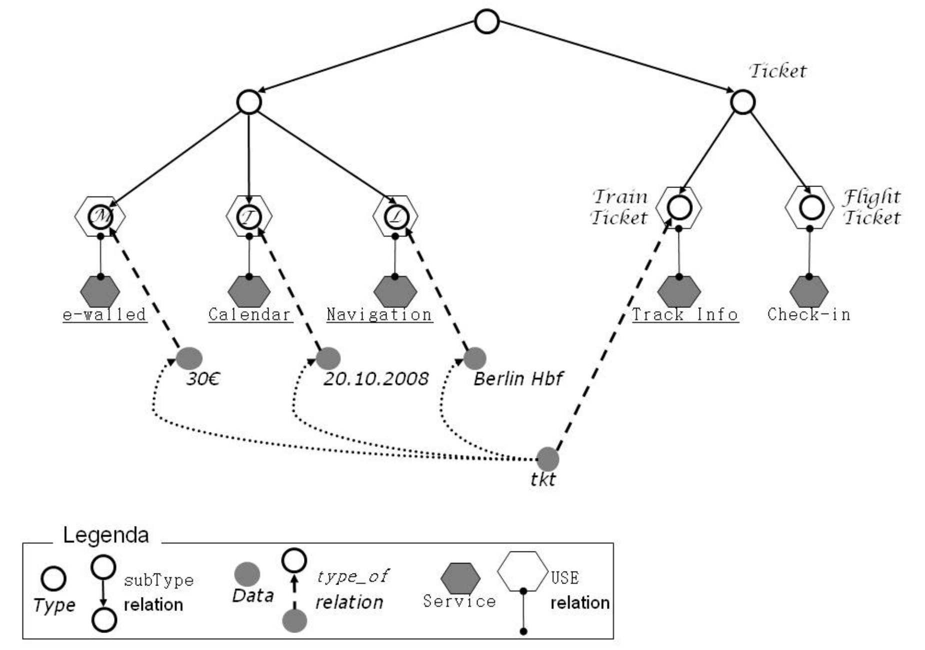

Fig. 1 A representation of service and objects in the ontology

ShoppingMall, where of course *US*, *Europe* and *ShoppingMall* are all restrictions or subtypes of *Location* that are supported by the ontologies used to represent the data processed by the services. Similarly, depending on the availability of type conjunction and disjunction it is possible to say that the navigation service can be used in for both *Location* and *Routes* meaning that it is good for both types of information.

3 Matching Algorithms

In Use-based discovery the matching process should satisfy two requirements: first, the discovery should be bidirectional; and, second, the discovery should support partial matching of objects and services. The bi-directionality means that the matching process should be triggered by both objects and services. Triggering the process from objects leads to the discovery of services which apply to a given object; while triggering the discovery from services leads to the discovery of objects to which a given service applies. The solution that we adopt is described in details in sections 3.1 and 3.2 below.

The second requirement, namely the partial matching of objects and services, relates services to only some of the aspects of objects. For example, in Fig. 1, the navigation system does not relate to the ticket as a whole, but only to its location component. To this extent, we need to compute an extension of the USE relation that relates services to some properties of objects. We compute this extension through an *association* function that relates the data to the types of its properties. The service discovery can then follow associations to find the different parts of the data to which services may apply.

To this extent, given some data *d* with properties $k_1,\ldots,k_n$, the set of types $A \subseteq T$ that can be associated to *d* (T being the set of all types), is defined by the following *associate* function:

a. $\forall t \in \mathtt{Type(d)} \Rightarrow t \in A$

b. *if* $d=<k_1,\ldots,k_n>, \forall k_i \in \{k_1,\ldots,k_n\} \wedge \forall t \in \mathtt{associate}(k_i) \Rightarrow t \in A$

Definition 1: Definition of *associate* function

The first condition specifies that a data instance *d* is associated with its own types; the second condition specifies that if a data has a set of properties $k_1,\ldots,k_n$, then it is associated to these attributes. As an example consider the train ticket *"user Ticket"* in Fig. 1, for a show at a given destination *"Berlin Hbf"*, at a given date "*20.10.2008*", with a given cost "*30 €*". Using the formalism outlined above we define:

user Ticket =< Berlin Hbf, 20.10.2008, 30 € >

Using the definition of *associate* above results in the following set.

associate(*tkt*)={*TrainTicket, Location, Time, Cost*}

Here *TrainTicket* has been added because it is the type of *tkt*, while *Location, Time* and *Cost* have been added because they are the types of the properties of *tkt.*

The example also highlights the difference between the classification and association. A train ticket is by no means a location, so it cannot be classified as belonging to the type such *Location*, but it is associated with such type by virtue of its *destination* property; equivalently, a train ticket is not a cost or a moment, but it is associated to *Time* and *Cost* through its properties *departure date* and *cost* properties.

3.1 Discovery Services from Objects

The objective of the algorithm presented in **Fig**. 3 is to find all the services that can be used in conjunction with some given data. This is the process that was highlighted in the previous example in which we found services for people and locations to be used in conjunction with appointments.

The algorithm is essentially a search across the taxonomy of types to find all services that can be utilised in conjunction with some given data. More precisely, line 1 specifies the data that is the input parameter, line 2 specify a variable to store the services found (**FoundList**); line 3 specifies the set of types from which to start the

```
1. Let $d$ the data to start from
2. Let FoundList=$\emptyset$
3. Let A = associate($d$)
4. For all types $t \in A$
5.   For all types $u$ such that $u \sqsupseteq t$ or $u \sqsubseteq t$ or $u \equiv t$
6.     For all services $s$ such that ($s$ USE $u$)
7.         if not filter($d$,$s$)
8.             then FoundList= FoundList$\cup\{s\}$
9. return FoundList
```

Fig. 2: Algorithm Relating Services to Objects

search: these are the types that are associated to the data ***d*** on the bases of the association function defined above. Line 5 starts the search through the type hierarchy. Line 6 selects all services that can be used to process data of type **t** by searching all its subclasses and superclasses; line 7 applies filters that can weed out services on the bases of different criteria; line 8 records the services found so far, and finally line 9 returns the list of services found.

It is easy to see that the algorithm always terminate when there is a finite set of types that are not recursive, since in the worse case all of them will need to be checked and none of them is checked more than once. To deal with the recursive types it is enough to keep track of the types crossed in the associate function, and make sure that no type is visited twice.

Line 5 and 7 may prove confusing and potentially controversial aspects of the algorithm. Specifically, line 5 directs the discovery to both the sub-types as well as the super-types of the type *t*. Indeed, in object oriented programming given a piece of data, the search for appropriate methods proceeds upwards toward crossing all super-types up to the root of the type tree and does not analyze all the methods available to the sub-types. The point to be noticed here is that the USE relation is looser than the type association in object oriented programming, therefore although the a service is specified for more specific data, there may be enough information in the data ***d*** to invoke the service anyway. Instead the filters specified in line 7 have been introduced as placeholders to enforce user customization. Such filters may select only services from a given provider, or it may provide only services that are proven to be of a given quality and so on. In this paper we do not concern ourselves with the definition of such filters, we just notice that they will be indeed required.

3.1.1 Example of Service Discovery

Referring back to Fig 1, above, the data of the user is represented through labeled gray circles, which are linked to the corresponding types through a `type_of` relation. In the example, the user purchases a train ticket represented by the label "*tkt*" which is specified of type *Train Ticket*. "*tkt*" contains three attributes: the cost which is "*30 €*" specified of type *Money*, the date of travel "*20.10.2008*" of type *Time*; and "*Berlin Hbf*" of type *Location*.

When the ticket is added to the data of the user's data, the above algorithm is performed and the first step is to compute the list of associated types to be stored in the variable **A**. Such computation is driven by the conditions specified in definition 1 above. As discussed above, the type *Train Ticket* is added to through the first condition of the definition, while the types *Money*, *Time*, *Location* are added through the second condition.

The next step in the algorithm is the search through the type system. In this example, such search is quite trivial since, for simplicity reasons, the type classification is underspecified. Nevertheless, through the loops defined in lines 5 and 6 of the algorithm the services `e-wallet`, `Calendar`, and `Navigation` are found. Under the assumption that none of them is filtered, they are added to the **FoundList** and returned as discovered services. The discovery of the services is represented in Fig. 1 by the underline under the service labels. Crucially, the service `Check-in` is not discovered because it is to be used with a type that is not recognized as associated with "*User Ticket*".

3.2 Discovery Objects for services

In mobile and ubiquitous computing, services may be local; therefore new services may become available when the user enters a new environment. An example of this type of services is the gate notification service at the Manchester airport that reports directly on the mobile phone gate that reports directly on the mobile phone gate information on the flights departing at the airport. Ideally, the discovery process should relate gate information to the airline ticket that is bound to be also stored in the mobile phone.

The algorithm shown in Fig 2 above finds the services that are relevant for the given data, but it fails to find to which data a new service can be applied. This problem is addressed by the algorithm shown in **Fig. 3** that maps a service to the available data exploiting both the `USE` relation and the "attribute" relation between data objects in the mobile phone. Therefore, when applied to the service at the Manchester airport, this algorithm should be able to relate the service to the flight tickets of the passengers.

More in details, the algorithm starts from a service **s** (line 1) that is in a `USE` relation with a type **u** (line 2) and establishes a storage variable **FoundList** (line 3). In Line 4 starts the search that for each super- and sub-types of the type **u** and for each data instance of those types (line 5). If the data is not filtered out (line 6) a call to the procedure `findAssociated` is issued to find the data that is associated with the type (line 7) and its results are added to **FoundList.** Finally **FoundList** is returned in line 8. The procedure `findAssociated` is the inverse of the associate function defined above. The filtering mentioned in line 6 are similar to the filters proposed in the algorithm in Fig 2: they are just placeholders for personalization filters that may be used in conjunction with the algorithm.

```
1. Let s be the service to start from
2. Let u such that  s USE u
3. Let FoundList = ∅
4. For all types t∈{u} ∪ super(u) ∪ sub(u)
5.   For all data d such that type_of(d,t)
6.      if not filter(d,s)
7.         FoundList = FoundList ∪ findAssociated(d)
8. return FoundList
```

Fig. 3: Discovery of services that match a given data

3.2.1 Example of Object Discovery

As an example of how the algorithm works consider a slight variation on the example presented in conjunction with Algorithm 1 above. In this example, the user has already purchased a train ticket to Berlin for the 20.10.2008 costing him 30€. Now consider two use cases. In the first use case, upon arriving at the train station, the mobile phone of the user detects that there is a service that reports track information (`Track Info` service), whose declared use is the type *Train Ticket*. Line 2 of the algorithm extracts the service use, namely *Train Ticket*. Line 4 and 5 perform the search for relevant data in the type *Train Ticket* or in its super and sub types. The *tkt*, being an instance of *Train Ticket*, is found among the data found. Assuming that *tkt* is not filtered out in line 6, a search for the data that is associated with it is triggered in line 7 through the invocation of `findAssociated`. Finally, in line 8, all data found will then be returned as candidate data for the newly found `Track Info` service. Therefore, the user can be communicated that a new service has been found, which is relevant for her train ticket.

In the second use case, assume that the user is provided with new calendar service that uses *Time* objects. In this case, the search for relevant data is equivalent to the previous case. Specifically, first the procedure looks for all data of type *Time* or of its super or sub types. As a result the date *20.10.2008* is found among possibly other data. Assuming that the data is not filtered in line 6, in line 7 the search for all associated data is triggered. This time `findAssociates` is invoked on the parameter *20.10.2008*. This function, pursuing the association in the opposite direction will return the set {*tkt*}. In this case, the new calendar can be offered to store the date of the train travel.

Crucially, in the second use case, the calendar is not associated to the time instance, but directly with the train ticket. This is because the train travel is the activity that the user intends to perform and the train ticket is the object that has the greater relevance to the user. At this time it is an empirical question whether any service should also be reflected in the time of travel.

4 Initial Evaluation

Whereas "use-based" discovery made sense in the scenarios that we were looking while developing our work, we run the risk that the overall framework works only under the idealistic assumptions that are made while developing a new way to think of a problem. Under those idealistic assumptions we cannot predict whether "use-based" discovery will fail to provide any interesting result.

To provide an initial evaluation of our ideas, we referred to the OWLS-MX [6] toolkit, which provides the description of more than 1000 services and the ontologies that are used to interpret those service descriptions. These services have been defined independently of our project, therefore they are challenging for our framework.

Table 1. Results of the experimental evaluation

	Ontology Metrics				Services / objects	Time
	Classes	Object properties	Data properties	Individuals		
1	2678	527	78	895	36	16
2	2673	527	78	860	18	12
3	2674	527	78	887	36	17

4.1 Performance Evaluation

First, we wanted to evaluate "quantitative" characteristics of the algorithm with respect to the OWLS-MX toolkit and the OWL-based reasoning. For this experiment we selected 81 services from different domains, relevant for our scenarios, since most of the services in OWLS-MX can hardly be deployed in pervasive settings. We have augmented the service definitions with the necessary `USE` descriptions, and also created a set of 37 data objects with complex structure to test the presented discovery algorithm. The algorithm implemented on the OWL-API 2.2.0. We conducted a series of experiments with different ontologies and services both for the discovery of services for objects and for the discovery of objects for services. The results of this quantitative analysis are summarized in Table 1 which shows the size of the domain, number of services/objects tested, and the total amount of time (sec.) to discover those objects and services. As it follows from the results, the algorithm is rather efficient.

4.2 Usability Evaluation

Second, we aimed at evaluating the "qualitative" applicability of the use-based service discovery approach. Indeed, while the description of service inputs and outputs is uniquely defined by the service signatures, the definition of service "use" may be rather subjective, and thus the discovery results may vary when the same services are advertisements are defined by different engineers. The goal of the evaluation was, therefore, to understand the "degree" of these variations, and to see how much the presented algorithm is able to abstract them away.

To perform such an evaluation, we split in two teams, one based at DOCOMO Euro-Labs in Munich, the other based at FBK-Irst in Trento, and we used the available ontologies to describe the selected services independently. Finally, we performed two different evaluations: the first one to analyze the differences between the descriptions provided by two groups, and the second one to match the selections of one team against the other using the implementation of the matching algorithms proposed above. With the first evaluation we measured the intuition about the "use" of a service. With the second evaluation we verified the ability of the matching algorithm to abstract the differences between different advertisements.

4.2.1 Analysis of Descriptions

With this evaluation we obtained the following results. Among the 81 service descriptions, 9 were equivalent (i.e., the same or equivalent concepts were used for specifying service advertisement), 22 were overlapping (i.e., the concept used by one group was more general than the other). In the remaining cases the advertisements used completely disjoint concepts. These numbers show that service advertisement is very subjective, and it is possible to advertise the same services in very different ways. Consequently, one could expect that objects will be mapped to completely different (and arbitrary) set of services. However, if we consider only those services that intrinsically refer to certain context (e.g., to locations), this number becomes much better: only for 21 services the description is disjoint.

In defining these descriptions we had to deal with two important problems. The first one was that the derivation of USE relation from the description of the service in many cases degrades to the input specification (e.g., relating parking service to the concept of "car" rather than to the location of parking).

The second problem was related to the ontologies loaded by OWLS-MX that do not describe user objects and user activities, and therefore were hardly applicable in our case. Furthermore, they make a very limited use of OWL properties, limiting the possible specializations. For example, it would be nice to claim that the service `Francemap_service.owls` (service that provides a map of France) is restricted to locations in France, but when we specified that the service can be used in conjunction with "SUMO:Geographic Region" or "SUMO:Translocation" it was impossible to restrict the scope of the service to France only. Such a lack of expressivity of the ontologies contained in OWLS-MX worsened the quality of the service representation, providing an intrinsic bias against our algorithsm.

4.2.2 Analysis of Discovery Results

We tested algorithm with both descriptions, matching the service description produced by one team against the service descriptions of the other team (FBK vs DOCOMO), and vice versa. As a result, we obtained the following outcome.

The precision and recall of the test was 0.63/0.54, and in only 8% of cases no services were found. One of the main reasons behind this result is that we considered not only the services and object that are related to specific user activities, but also generic services and object that were unrelated and created considerable noise.

In a second measurement, we restricted to objects and services that correspond to certain context and activities and the results improved: the precision and recall rose to 0.73/0.55. Furthermore, even if the service advertisement specifications do not match, the discovery algorithm manages to smooth down the differences between the service advertisements. The reason of this abstraction is that the relation between services and objects is based on the properties of these objects, which lowers the dependence on the specific advertisement.

The experimental results show that the proposed approach is indeed applicable in the scenarios and domains like those discussed in this paper: where pervasive services are explored and where objects associated to the user activities are considered. In addition it shows that the more structured objects are, the better discovery results may be obtained.

5 Discussion

Use-based discovery, as discussed so far has a form of advertisement that is completely unrelated from the description of the service as a computation entity. In this sense, it is very different from our Web services languages, such as OWL-S [7], SAWSDL [2] or WSML [10] which directly ground on computational features of the service like Inputs, Outputs, message formats and so on. On the opposite of what can be done with these languages, a service can be declared to be useful for all sort of objects, even though such definition is totally meaningless.

The lack of direct relation with the service has both positive and negative effects. On the positive side, the `USE` relation can be used to describe any information source, and therefore it is not limited to services. Indeed one could specify the expected use of Web pages, and of RSS feeds. Indeed, the user train ticket in Fig. 1 may be related to a Web page reporting the train schedule, or a RSS reporting news about the rail system, expected delays and so on. Ultimately, Use-based discovery contributes to bridge the gap between Web services and the rest of the Web.

On the negative side, the USE relation is too subjective; therefore the same service may be described in very different ways. Furthermore, the USE relation does not provide any information about the invocation of services: ultimately the user may know that a service relates to a given object, but she may still have the problem of invoking the service.

In our current work, we are trying to address these two problems. Specifically, the problem of handling invocation can be addressed by defining lifting and lowering functions that given a USE description describe how to generate the service inputs and interpret service outputs. These functions are described in details in section 5.1. As for a more precise definition of the relation between USE and services, we are evaluating different approaches that can be used to address the problem of tightening the relation between the service and the description. These ideas will be discussed more in details in section 5.2.

5.1 Exploiting USE for Invocation

As pointed out above, the `USE` relation specifies what a service can be used for, but it does not specify how to invoke the service, therefore it does not provide any guidance to the user in the actual use and invocation of the service. To address the problems listed above, the `USE` specification needs to be supplemented with invocation information. Although the service invocation is beyond the scope of this paper, the relation to the invocation information can be performed by leveraging on existing solutions such as OWL-S [7] and SAWSDL [2]. Specifically, the `USE` specification can be mapped to a workflow language that specifies which operations of the service should be performed to realise a given use of the service and in which order they should be performed. The relation between the `USE` specification and the workflow language is equivalent to the relation between the OWL-S Profile and the OWL-S Process Model, whereas the Profile specified the capabilities of the service, and the Process Model how those capabilities are realised by the service.

In addition, the specification of the `USE` of a service needs to relate to the inputs of the service, so that the given the use it is possible to generate the inputs of the service itself. This relation was specified in OWL-S [7] by using in the Profile the input and outputs of processes specified in the process model. The OWL-S solution of course is not possible in the context of the discovery proposed in this paper since the `USE` specification is not based on the inputs and outputs of the service. Instead, we adopt the solution proposed by SAWSDL [1] that proposes the use of *lowering functions* to map the inputs and outputs of operations to the corresponding concepts in the ontology. Therefore, the `USE` specification should be enriched by lowering functions that specify how the data specified in the use can be mapped in the data needed by the operations that the user wants to do with the service.

Specifically, given a service `s`, let I be the inputs of service `s` and $\mathcal{U}$ be the specified usage for `s`; we define the lowering function for `s`, $\texttt{lower}_{s}\colon \mathcal{U}_s \rightarrow I_s$, which specifies how data that is defined consistently with the use of `s` ($\mathcal{U}_s$) can be transformed in the set of inputs of `s` (I_s) required by the service. For example, given an appointment, the lowering function may specify how to extract the location and format it in a way that can be processed by a routing service. Crucially, when such a lowering function is complete, then the usage specification $\mathcal{U}_s$ guarantees that the service client has all the information that is required to invoke the service.

In addition to a lowering function, consistently with SAWSDL we need to specify *lifting* functions that how the outputs of the service reflect in the ontology. Formally, the lifting functions is defined as $\texttt{lift}_s\colon O_s \rightarrow C$, where O_s are the outputs of the service, while C is a concept in the ontology. The definition of the lifting function highlights an asymmetry with lowering functions. Whereas, lowering functions need to relate the service directly to the service specification because at the time of invocation the service client needs to map the use into the inputs of the service, the lifting functions may result in the specification of data items that may be unrelated to the specified use. An example of this behaviour are recommendation services, for example a movie recommendation service may be called in conjunction to a movie title or a movie ticket, yet, it does not extend the definition of either the movie title or the movie ticket.

As an example of using the lifting and lowering functions consider the use of the calendar service. The calendar service may be advertised to be used in conjunction with data of type $\mathcal{Time}$; furthermore, the service may provide the typical three operations that associated to calendars: `record`, `view` and `delete`. Furthermore, `record` may take a time instance as input and return a record number; `view` may return a record number and display the results on the screen; while `delete` may take a record number and remove the corresponding record from the calendar.

The `USE` of the calendar service may be specified as being objects of type $\mathcal{Time}$. Once the `USE` is specified, the lifting and lowering functions for the different operations can also be specified. The lowering function for `record` map $\mathcal{Time}$ into the format that is required by the record operation, while the lifting function maps the record number either in a new object or in a new property to be added to a specialisation of $\mathcal{Time}$. Similarly, the lowering for `view` and `delete` take the inputs objects that contain a calendar record number and map to the inputs of the respective functions. Finally, the lifting functions for the two operations may not be interesting since given the description above neither operation return any value.

The use of lifting and lowering functions provide one of the ways to guide the invocation and to filter results of the matching algorithms described below. Considering the example in the paragraph above, it can be noted that until a record number is generated there is no way to use the `view` and `delete` operations, therefore the invocation of such operations on a given object will be delayed until the `record` operation is performed. Similarly, services for which the input cannot be filled by the data available to the client can be discarded.

5.2 Improving `USE` precision

To improve the precision of the USE relation, we need to tighten the relation between a service and its specified use. To achieve this goal, we are following two different approaches. The first one attempts to weaken the original statements on which `USE` was based. Namely, that `USE` is independent of inputs and outputs of the service. In this sense, we could exploit the lifting functions described above to figure out what inputs the service requires, and then relate these inputs to objects through the relation of these concepts to the existing objects in the ontology. From an implementation point of view, this process can proceed along the “associate” links described in section 3. While this approach may prove able to describe the relation between services and objects to which they relate, its weakness is that it is not able to describe Web pages and RSS feeds that could be useful for the user.

A second approach is to relate services, web pages and objects with the concepts representing the “resources” that they manipulate. The idea of relating services and objects such as tickets with resources is described in [5] with the idea of pushing the service composition as a whole. In this approach the USE relation will then be inferred by the declarations of both services and objects, and likely it will not be as generic as it is defined so far.

6 Conclusions

The service discovery mechanisms that have been proposed in literature fail to address the problem of discovering the pervasive services that are emerging in the mobile environment The reason of this failure is that discovery is thought as a goal directed process in which the service is found because it achieves (part of) the goals of the requester. As a consequence service discovery concentrates on the technical aspects of the services, such as its inputs and outputs trying to derive from them the potential use of the service.

On the opposite of this view, discovery in pervasive service provisioning is not goal directed, rather services associate to the activities of the user adding value and information to the data that the user needs to utilize. To address this different use of services, in this paper we proposed "use-based" discovery: a framework in which the expected use of a service is specified, and then this use is matched against the data that to the services. To evaluate this idea, we tried to describe 81 services from the OWLS-TC testbed and we showed that even if the service advertisement strongly depends on the requirements and objectives of the provider of the description, the proposed algorithm is able to smooth this aspect and to deliver reasonable enough set of services that correspond to the user objects and activities.

7 References

1. Akkiraju R., Srivastava B, Ivan A., Goodwin R, Syeda-Mahmood T;Semantic Matching to Achieve Web Service Discovery; Conference on Enterprise Computing, 2006
2. J. Farrell*, and* H. Lausen "Semantic Annotations for WSDL and XML Schema -W3C Recommendation 28 August 2007*" Available at* http://www.w3.org/TR/sawsdl/*; Last viewed 1.October 2008*
3. Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu, Shivaun Albright; "Simple Service Discovery Protocol/1.0
4. Operating without an Arbiter";Internet Engineering Task Force, INTERNET DRAFT
5. Raman Kazhamiakin, Piergiorgio Bertoli, Massimo Paolucci, Marco Pistore, Matthias Wagner: Having Services "YourWay!": Towards User-Centric Composition of Mobile Services. FIS 2008: 94-106
6. Klusch, M.; Fries, B.; Sycara, K. (2006): Automated Semantic Web Service Discovery with OWLS-MX. Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, ACM Press.
7. D. L. Martin, M. H. Burstein, D. V. McDermott, D. L. McGuinness, S. A. McIlraith, M. Paolucci, E. Sirin, N. Srinivasan, K. Sycara; Bringing Semantics to Web Services with OWL-S; World Wide Web Journal 2006
8. *C. Enrique Ortiz;* "An Introduction to Near-Field Communication and the Contactless Communication API"; available at http://java.sun.com/developer/technicalArticles/javame/nfc/#11 (lv: 25.06.2009)
9. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic Matching of Web Services Capabilities," in Proceedings of the 1st International Semantic Web Conference (ISWC2002)
10. Dumitru Roman, Holger Lausen, and Uwe Keller; "Web Service Modeling Ontology (WSMO)"; http://www.wsmo.org/TR/d2/v1.2/ (lv 25.06.2009)

11. K. Sycara, S. Widoff, M. Klusch and J. Lu, "LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace." Autonomous Agents and Multi-Agent Systems, 5, 173–203, 2002.
12. K. Sivashanmugam, K. Verma, A. Sheth, J. Miller; Adding Semantics to Web Services Standards; The 2003 International Conference on Web Services (ICWS'03); 2003
13. M. Hamdy, B. König-Ries, U. Küster; “Non-functional Parameters as First Class Citizens in Service Description and Matchmaking - An Integrated Approach”; in Proceedings of NFPSLA-SOC 2007
14. G. Zhou, J. Yu, R, Chen, H. Zhang; Salable Web service discovery on P2P overlay; Service Computing Conference, 2007