Context-aware Semantic Service Discovery

Vincenzo Suraci[1], Silvano Mignanti[2], Anna Aiuto[3]
University of Rome "Sapienza", Department of computer and system sciences
[1]vincenzo.suraci@dis.uniroma1.it, [2]silvano.mignanti@dis.uniroma1.it, [3]aiutanna@libero.it

Abstract – **In the last few years telecommunications and internet have spread all over the world, in a pervasive way, connecting millions of devices, people, sensors and services without a planned strategy. In such scenario the discovery of services represent still an open challenging research field.**

To address that problem this paper proposes a context-aware semantic service discovery architecture designed to perform distributed Service Discovery in heterogeneous networks. This novel architecture is technology independent and compatible with most of the existent service discovery protocols; it inherits and extends the results of the last research groups in the field of context-aware service discovery based on the use of semantic languages.

The present work presents the first results of the service discovery design activity which has been carried out within DAIDALOS II (Designing Advanced network Interfaces for the Delivery and Administration of Location independent, Optimised personal Services), a project granted in the European 6th Framework Research Programme, within the IST (Information Society and Technology) thematic area.

Index Terms – **Service Discovery, Context awareness, Semantic Description, Service Ontology, OWL-S.**

I. INTRODUCTION

Recent improvements in embedded system technologies and the increase of Broadband Internet accesses have caused a dramatic spread of interconnected people and services. In such pervasive computing environments, where a great number of sensors and devices collaborate and provide numerous services, a powerful service discovery is a vital process.

Current service discovery protocols, such as Jini, UPnP, Salutation, UDDI do not make use of contextual information in discovering services, and, as a result, fail to provide the most appropriate and relevant services for users [11], [12]. In addition, current protocols rely on keyword-based search techniques and do not consider the semantic description of services. Thus, they suffer from poor precision[1] and recall[2]. This leads to the possibility of using context to improve service discovery.

The context-aware computing literature [4] defines context as "information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to interaction between a user and an application including the user and application themselves". Context awareness, on the other hand, is defined as "a property of a system that uses context to provide relevant information and/or service to the user, where relevancy depends on the user's task" [4]. Consequently, context-aware service discovery can be defined as the ability to make use of context information to discover the most relevant services for the user.

To be able to provide Context-aware services (CA services), these need to consider context input besides functional[3] input. Therefore, the output of context-aware service now also depends on contextual information. Contextual inputs are provided by context providers that form a new party in the service provisioning process. Besides context inputs, a service exhibits context itself. Services that exhibit context but do no require context inputs are not considered context–aware, because they do not use user context to provide tailored information. However, they can be context-aware discovered.

Various models have been proposed to represent context information: for example, attributes and values models, mark-up language models, first order logic models, object oriented models and ontologies.

Ontologies have proved to be the most suitable model [8] for representing and reasoning on context information for the following reasons: (i) ontologies enable knowledge sharing in open, dynamic systems; (ii) ontologies with well defined declarative semantics allow efficient reasoning on context information; and (iii) ontologies enable service interoperability and provide the means for networked services to collaborate in a non ambiguous manner.

For describing the semantics of services, the latest research in service-oriented computing recommends the use of the Web Ontology Language for Services (OWL-S) [2]. OWL-S is an ongoing effort to enable automatic discovery, invocation, and composition of web services. Even though OWL-S is tailored for web services, it is rich and general enough to describe any service. Furthermore, OWL-S does not include a semantic description of contextual information. Thus, in its current state, it does not support context aware discovery in pervasive computing environments.

[1] Precision – a standard measure of information retrieval performance, defined as the number of relevant items retrieved divided by the total number of items retrieved. The highest value of precision is achieved when **only** relevant items are retrieved [9].

[2] Recall – a standard measure of information retrieval performance, defined as the number of relevant items retrieved divided by the total number of items in the collection. The highest value of recall is achieved when **all** relevant items are retrieved [9].

[3] Functional inputs refer to the typical service's properties, such as Inputs, Outputs, Preconditions and Effects (IOPEs).

The paper proposes a context-aware semantic service discovery architecture using an innovative context aware filtering process. Section II describes the ontology–based Service Description languages; while section III presents the state of the art in Context Aware Service Discovery. Section IV and V present, respectively, our reference context and discovery architecture. Section VI describes the registration and discovery functionalities; and section VII presents an OWL-S extension to share a common ontology. At the end some considerations point out the innovations introduced by the present work.

II. Ontology Based Service Description

Most of the existing service discovery protocols retrieve services descriptions that contain particular keywords from the user's query (SLP, UPnP, Bluetooth, etc.). In most of the cases this leads to low recall and low precision of the retrieved results. The reason for low recall is that query keywords might be syntactically different from the terms in service descriptions; while low precision is due to the fact that query keywords might be syntactically equivalent but semantically different from the terms in the service description. Thus, traditional service discovery mechanisms have limited quality results, because they lack a common understanding on services. Matchmaking based on different knowledge levels is hard, if not even impossible. Therefore, ontology is an envisioned mechanism to overcome this limitation.

Ontologies enable semantic matchmaking; they define concepts and their relations that specify semantics of information better than non-related keywords. These semantic definitions can be used to match service and request based on properties and their semantics rather than on keywords (e.g. syntactic matchmaking). This can lead to more relevant discovery results. The main goal of ontologies is to create a common understanding on a specific domain. By creating this common understanding, the sketched problems with keyword-based service discovery mechanisms can be overcome.

A lot of research activities have been carried out to define ontology languages to be used to describe the semantic web services. A Semantic Web Service results from the combination of the Semantic Web and Web Services. The Ontology Web Language for Services (OWL-S) is the most complete effort for describing semantic Web services. Besides describing high-level capabilities of services, OWL-S allows the description of services' behaviours using conversations.

OWL-S defines Web services capabilities in three parts: the Service Profile, the Process Model and the Service Grounding. The Service Profile gives a high-level description of a service and its provider; it is generally used for service publication and discovery. The Process Model describes the service's behaviour as a process, either atomic or composed. The Service Grounding specifies the information necessary for service invocation, such as underlying communication protocols, message formats, serialization, transport and addressing information. The Service Grounding defines mapping rules to link OWL-S atomic processes to the web service description language (WSDL) operations.

In the present work we will extend the OWL-S capabilities in order to cooperate with a semantic knowledge base expressed in OWL [3], shared between all the components of our architecture.

III. Latest Research Activities on Context-Aware Service Discovery

In the Context-Aware (CA) service discovery, an association between the service requestor, service provider and context provider needs to be created.

There are two major reasons why the matching process can initiate the search for context:

1. *Request completion*: applied to client side, it should be performed when the service requestor wants to find services based on contextual criteria; this information is used to match the request with the context of the service and of the surrounding environment. By completing the request with contextual information on the requestor and with context requirement on the service and the environment, the precision of the results that are more relevant can be returned.
2. *Input completion*: applied to service provider side, it should be performed when possible services require context inputs that the service requestor or the surrounding environment does not provide. By completing missing inputs, the recall of the result is improved since the services that would be ignored when not using context, are now returned.

The context-aware service discovery can play an important role in improving the quality of the discovery result: recall and precision rates become higher when including context in the service discovery process.

On the one hand, the contextual information makes the user's query more information-rich and, thereby, provides means for high precision of the retrieved results. On the other hand, the contextual information can serve as an implicit input to a service that is not explicitly provided by the user. This prevents filtering out the services that require this input from the user, which leads to higher recall of the retrieved results.

The nature of context introduces some limiting factors on how well precision and recall can improve. When the context is imperfect, the precision and recall decrease because incorrect information is used. Furthermore, alternative representations, among the service requestor, providers and context providers, results in mismatches and decreases precision and recall.

Some system requirements for CA service discovery mechanisms can be derived. First of all a *Common understanding* is required. In fact, by introducing context, another party is introduced in the service provisioning process: the context provider. Service provider, service requestor and now also context provider have different knowledge of the services and without mechanism to create a common understanding, poor discovery result may occur. On the other hand a *Semantic matchmaking* is necessary: context has different representations making them semantically rich. This increases the need for semantic description mechanisms and semantic reasoning.

In the light of the above description, the Context-aware, Ontology based, Semantic Service Discovery (COSS) [10] - Service discovery approach is characterized by three major aspects: (i) context-awareness, (ii) ontologies and (iii) semantic reasoning. COSS collects and uses contextual information to complete required inputs for services that the user does not explicitly provide; it uses the Ontology Web Language (OWL) to create common domain specific understanding; and it makes use of the ontologies semantic reasoning, even if it reasons only the *subclass* relation. The real limits of this architecture are the use of domain specific ontologies and the impossibility for the user to weigh the properties he likes the service to have. These aspects could improve the quality of the service discovery result. Furthermore, as mentioned before, this approach supports only the OWL *subclass* relation, which is a transitive relation; in reality, there are many more types of relations.

Another solution is the Context-Aware Service Enabling (CASE) platform [7] that combines context-aware service discovery with service composition. The main function of the CASE platform is to dynamically adapt services by changing their composition in response to context changes. To accomplish this, the platform consists of a composition service and two types of discovery services: a context aware discovery service and a basic discovery service (based on a Jini lookup service). Context-aware discovery services, other services in the Jini network, and context sources are all Jini services. Context information is obtained through context agents, each one storing references to context sources that can provide context information about the associated entity. Both active and passive discovery are supported. This approach uses ontologies to describe context sources and services. The main disadvantage of CASE approach is to lay on a specific service discovery protocol, Jini, which is limited to a local area network environment and to a specific programming language.

The architecture proposed in [1] represents both the contextual information and service description, using an ontology-based approach, a shared ontology that is an extension of the Standard Ontology for Ubiquitous and Pervasive Application (SOUPA [5]) in order to represent additional contextual information. It defines and represents generic concepts, such as person, agent, time, space, and event; but it lacks a clear semantic description of services: thus, it has been coupled with OWL-S. The combined ontology is shared among all the entities in the environment, including the context engine, the discovery component, services, users, and providers. The main lack of this approach is an integrated architecture completely based on the power of OWL (to manage the context) and OWL-S (to manage the service description), which is the main effort of the present work.

IV. Reference Context Architecture

We have adopted a totally different reference context architecture where it is possible to distinguish the following entities (see fig. 1):

- The *User*, that is the entity starting the discovery process and receiving the list of service found;
- The *Service*, that is what the end user is looking for;
- The *Environment*, that is the pervasive entity in which the user is involved;
- The *Context Manager*, that is the entity which administrates, stores, retrieves and distributes context information

User, Service and Environment entities produce context information by means of proper sensors. These sensors are interfaced with the context manager through the context sources (CS). For example, an end user equipped with a GPS-enabled PDA can provide some context information, like the terminal battery charge, his geographical position, the terminal screen size, etc., using proper software or hardware sensors. The raw context data are computed by the context source in order to be sent and stored in the context manager (CM). The role of CM is to store the context information produced by the context sources. It also provides a Publish-Subscribe mechanism to make the context information be available to the external applications that need it. Finally, it also provides some filtering rules to select the required context information.

A generic service produces context information. A fax service, for example, is characterized by an outgoing queue length, the cartridge status, the number of remaining sheets and so on. Also the pervasive environment in which the user is involved in can provide useful context information, like temperature, connectivity, pollution, presence detection and so on.

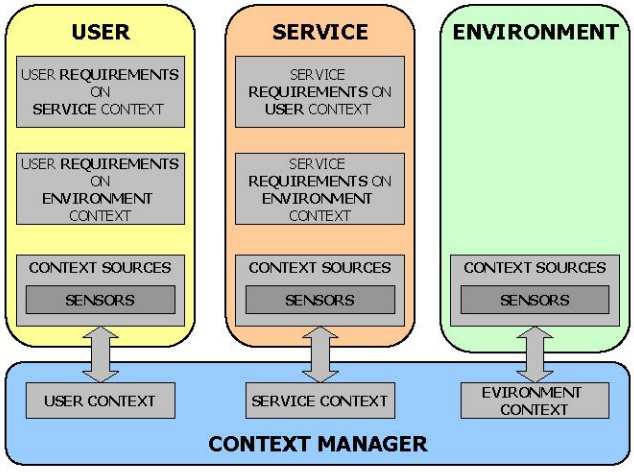

Figure 1 – Context Architecture

On the other hand, user and service entities have some requirements on the context information they need in order to work properly. An user can specify some requirements on the context of the services he is looking for, like its availability status or location, and on the context provided by the environment, like the availability of a wireless connection or the presence of air conditioned. A service can require the user to provide specific context information, like its location or some terminal capabilities, and the environment to provide context information, like movement detection or network quality of service.

V. Reference Discovery Architecture

Nowadays almost all the Service Discovery Protocols are based on a simple framework including three main

entities: (i) the *User* which triggers the discovery process sending a *basic query* to (ii) the *Service Directory Server,* which stores the *basic service descriptions* in a *registry*, and *filters* them on the base of the basic query, replying to the user the list of available services matching the search criteria. (iii) The *Service Provider* communicates with the Service Discovery Server to publish the services it owns.

In order to achieve backward compatibility to the existing discovery architectures, we extended this basic discovery mechanism introducing an integrated solution to deal with a semantic context-aware discovery process. The reference discovery architecture is shown is figure 2.

As already explained in the previous section, the user takes care about its own context thanks to the presence of CSs that point to the user context stored in the CM. The user also has some requirements on the context of the services he is looking for and the environment he is involved in. These requirements are expressed using semantic query languages, like RDQL or OWLQL. The user starts the service discovery process providing the service discovery server a query composed by two parts: the *basic query,* which is expressed in one of the low level discovery languages (like SLP, UPnP, UDDI, etc.), and the *semantic query,* which is expressed using a high level semantic query language.

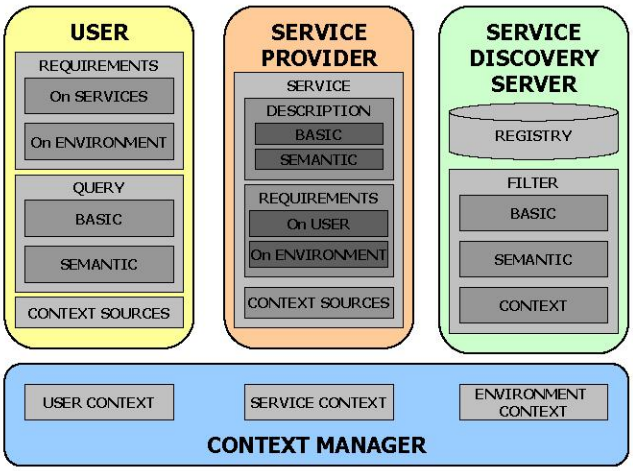

Figure 2 – Reference Discovery Architecture

The Service Provider (SP) owns one or more services. Each service has its own context stored in the CM, its requirements on the context of the user and the environment, which are expressed using a semantic query language, and its own description. The service description is represented by a basic description and a semantic description. The former is expressed using the description model of the low level discovery language (for example, in SLP, it is a list of attribute-value couples; in UPnP, it is an XML file); the latter is expressed using a semantic language (for example OWL or WSMO).

The Service Discovery Server (SDS) provides two main functionalities: it stores the service descriptions in a registry and it filters the available services using the information sent by the user.

VI. Service Registration and Discovery

Let's see now how the entire architecture works describing in detail the two main processes needed to find a service: the registration of a service and the discovery of a service. In order to register a service, the SP should:

- register the CSs of that service in the CM;
- define the requirements on the user and the environment for that service;
- define the service basic and semantic description:
- register in the SDS registry the service description, the service requirements and the pointer to the service context.

On the other hand to discover the needed services an User should:

- register its own CSs in the CM;
- specify the user requirements on the service context and on the environment context;
- specify the basic and the semantic query
- send to the SDS filter the query, the user requirements and the pointer to the user context.

Once the SDS receives the request from an user, it activates the filtering engine. One of the innovations introduced in this work is the three phases filtering mechanism performed by the SDS.

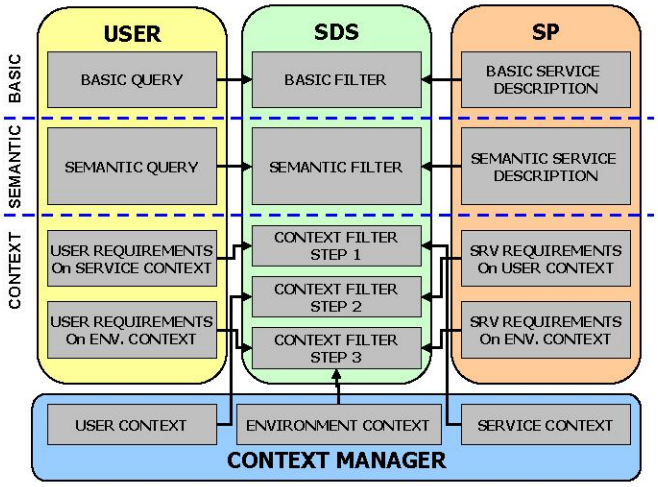

Figure 3 – Three phases filtering mechanism

The *Basic filter* involves the basic query of the user and the basic descriptions stored in the registry. For each description, the query is evaluated using a protocol specific filtering engine (i.e. the one of SLP or UPnP). This first filtering step is mainly used to perform a rough filtering, aiming to select the correct type of service and decrease the number of matching services to be passed to the next phase, which is more time consuming.

The *Semantic filter* requires the semantic query of the user and the semantic description of each service not filtered during the first phase. This step increases the recall rate and the precision of the overall filtering mechanism, thanks to the use of a semantic engine. The result is a list of services having exactly the characteristics the user wants, but without any restriction on the context information.

Finally a *Context filter* is applied. It consists of three sub-steps. First of all, the SDS gathers from the CM the context information of the service and then applies on it the User

requirements on the service context. Then the SDS retrieves from the CM the context information of the User, and applies on it the service requirements on the user context. Finally, the SDS gets (always from the CM) the environment context and applies on it both the user and the service requirements on the environment context.

VII. OWL-S Extension

The Reference Discovery Architecture has been designed to be independent from the specific low level service discovery protocols and the specific semantic languages used to describe the service descriptions and the context information. But an integrated ontology structure could be used in order to unify the semantic language used for the context and the one used for the service description.

The latest researches on this topic (i.e. [1], [5], [6], [8], [10] and [13]) show that the use of OWL-S for the description of services is an effective solution, but it does not include a semantic description of contextual information, thus a proper extension is needed.

The *ServiceProfile* class is enriched with a *context* attribute, which is a URL pointing to the real service context stored in the CM as an OWL file. In that way the whole architecture could simply lay on a shared OWL-based ontology where the service description follows the OWL-S rules, while the context is directly described in OWL.

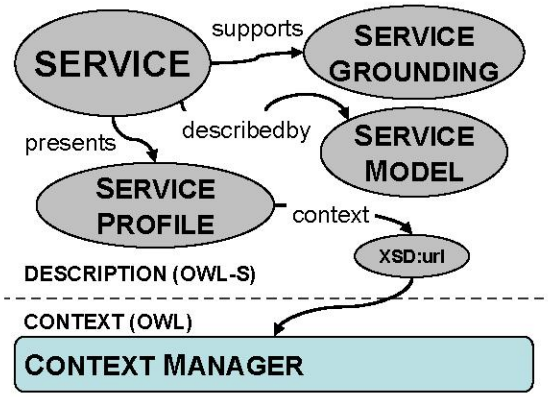

Figure 4 – extended OWL-S ontology

This solution is suitable but also elegant from a theoretical point of view. In fact the service profile represents the description of the whole service that includes not only the static service characteristics but also the dynamic service characteristics: the context. While the former can be easily stored in the SDS registry as a static OWL-S file, the latter changes dynamically and needs a CM to manage it. Whenever a context filter needs the service context information, it queries the CM using the *context* attribute.

Finally using this simple extension, the semantic and the context filtering engines lay both on the OWL filtering engine, simplifying the overall architecture complexity.

VIII. CONCLUSIONS

In this work we presented a context-aware semantic service discovery architecture able to give a concrete response to the main challenging issues risen up by the latest researches in this field. An effective context architecture model has been designed to be used by a technology and language independent service discovery architecture. A novel three phase filtering process has been introduced. Finally, an integrated semantic approach based on OWL has been presented in order to deal at the same time with the service description and the context information model.

DAIDALOS DISCLAIMER

The work described in this paper is based on the results of IST FP6 Integrated Project DAIDALOS II. The project receives research funding from the European Community's Sixth Framework Programme. Apart from this, the European Commission has no responsibility for the content of this paper.

REFERENCES

[1]. A.R. El-Sayed and J. Black, "Semantic-Based Context-Aware Service Discovery in Pervasive-Computing Environments".

[2]. W3C, "Web Ontology Language for Services (OWL-S) 1.1," http://www.daml.org/services/owl-s/1.1/2005.

[3]. W3C, "Web Ontology Language (OWL)." http://www.w3.org/TR/owl-features/, February 2004.

[4]. A. Dey, D. Salber and G. Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications". In Human-Computer Interaction, volume 16, pages 97-166, 2001.

[5]. H. Chen, F. Perich, T. W. Finin, and A. Joshi, "Soupa: Standard ontology for ubiquitous and pervasive applications". In MobiQuitous, pages 258-267. IEEE Computer Society, 2004.

[6]. X. Hang Wang, D. Q. Zhang, T. Gu and H. K. Pung, "Ontology based context modelling and reasoning using owl". In PerCom Workshops, pages 18-22, 2004.

[7]. C. Hesselman, A. Tokmakoff, P. Pawar and S. Iacob, "Discovery and Composition of Services for Context-Aware Systems".

[8]. S. Ben Mokhtar, D. Fournier, N. Georgantas, V. Issarny, "Context-Aware Service Composition in Pervasive Computing Environments", INRIA Rocquencourt 78153 Le Chesnay, France, http://www-rocq.inria.fr/arles/

[9]. L. Steller, S. Krishnaswamy and J. Newmarch, "Discovery Relevant Services in Pervasive Environments Using Semantics and Context".

[10]. T. Broens, "Context-aware, Ontology based, Semantic Service Discovery". Master's thesis, University of Twente, Enschede. The Netherlands, 2004

[11]. M. W. M. Feng Zhu and L. M. Ni., "Service Discovery in Pervasive Computing Environments", Pervasive Computing, vol. 4, no. 4, pp.81-90, 2005.

[12]. M. K. Steve Cuddy and H. Lutfiyya, "Context-Aware Service Selection Based on Dynamic and Static Service Attributes", in Proc. of the IEEE Int. Conf. On Wireless and Mobile Computing, Networking and Communications, (Montreal, Canada), pp. 13-20, August 2005.

[13]. D. Preuveneers, J. V. den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. De Bosschere, "Towards an extensible context ontology for ambient intelligence". In EUSAI, pages 148-159, 2004.