Discovering Relevant Services in Pervasive Environments Using Semantics and Context

Luke Steller, Shonali Krishnaswamy and Jan Newmarch

Faculty of Information Technology, Monash University,
Peninsula Campus, Melbourne, Australia
{Luke.Steller, Shonali.Krishnaswamy,
Jan.Newmarch}@infotech.monash.edu.au

Abstract. Recent advances have enabled provision and consumption of mobile services by small handheld devices. These devices have limited capability in terms of processing ability, storage space, battery life, network connectivity and bandwidth, which presents new challenges for service discovery architectures. As a result, there is an increased imperative to provide service requesters with services which are the most relevant to their needs, to mitigate wastage of precious device capacity and bandwidth. Service semantics must be captured to match services with requests, on meaning not syntax. Furthermore, requester and service context must be utilized during the discovery process. Thus, there is a need for a service discovery model that brings together 'semantics' and 'context'. We present a case for bringing together semantics and context for pervasive service discovery by illustrating improved levels of precision and recall, or in other words increased relevance. We also present our model for integrating semantics and context for pervasive service discovery.

1 Introduction

Recent years have seen a rapid increase in the usage and availability of small handheld wireless devices, which can provide or request services. These devices are far more heterogeneous and resource constrained than traditional desktop PCs. There is therefore an increased need for transmission of only relevant data and reduced processing costs. In addition, mobile users time constraints, given the highly dynamic and volatile operational environment they work in. Therefore, pervasive discovery architectures must return results containing only the most relevant services as deemed by the requester.

Current pervasive service discovery architectures utilise simple string attribute based techniques or interface comparison which fail to adequately discover the most relevant services. Capturing behaviour [1, 2] would allow requests to be matched with services that may be described differently but are semantically equivalent in meaning. It would also enable partial matching in the absence of an exact match. The descriptions must be rich enough to allow reasoning on the semantics to infer additional information from that provided, by exploiting logical relations between the concepts used. For instance, if a user wishes to know whether device A was in room

B at time T, this can be inferred from the knowledge that device A was in room C at time T, where room C is not physically located within room B.

W3C has defined semantic web service languages such as DAML-S/OWL-S [3] which meet these requirements. They are used to construct ontologies, which provide an abstract model, defining concepts and properties which relate to a domain of interest [1] and a hierarchy of relations between these, using a set of axiom rules. This is sufficient to define formal semantics of service behaviour that can be reasoned about.

In addition, mobile users seek services that are most relevant to them and their current situation. They generally prefer services that are from nearby locations, return fresh up-to-date, newly created data and return results that are displayable on the requester device [4]. Due to an increased freedom of mobility, the user's environment, location and the objects around the user are more dynamic. This necessitates the collection and use of context during discovery. Context-aware discovery demands the use of implicit information pertaining to both requester constraints and provider requirements, which can affect the usefulness of the returned results [5].

The interpretation of context information, which is highly interrelated and imperfect, can be influenced by perspective. It can be misleading if used out of context or without relating it to the domain of interest. Therefore, context must be described semantically so that valid deductions can be drawn from it. For instance, displaying something to nearest display may not guarantee the user can actually see the display; it might be behind a wall. Current architectures to date do not bring semantics and context together, to enrich the service matching process during pervasive service discovery.

In this paper, we demonstrate, through an illustrative case study, how matching services with requests based on semantic meaning rather than syntax, while also including context information, will improve the relevance of the pervasive services returned to the requester. We also propose a preliminary conceptual model which will support this assertion.

The remainder of this paper will be as follows: Section 2 reviews related work, section 3 presents our motivating scenario, section 4 presents our preliminary architectural model and section 5 concludes the paper.

2 Related Work

In this section we review related research and evaluate their support for rich expression of semantic meaning and context-aware discovery. Table 1 presents a comparison of these architectures.

In terms of semantics, Jini [6], UPnP [7] SLP [8, 9] and Salutation [10], Konark [11] and SSDM [12] use either interface of string based matching. Anamika [13] and MobiShare [4, 14] utilise RDF or OWL ontological service types to support limited subclass relations. They do not consider other relations such as 'part-of' and service type, alone, provides a category but leaves out the detail required to establish what the service does. DReggie [15] utilises DAML and supports reasoning and approximate matching. However, it is limited to a single predefined ontology which defines characteristics such as service mobility and service requirements, only. COSS [1] supports service type, inputs and outputs, approximate matching and ranking. However, it does not allow definition attributes beyond this and does not support reasoning.

In terms of context, both UPnP and Konark support state change events and LUDS [16] extends these for use in discovery, based on string comparison. SSDM and Jini Context only support easily rankable quality of service (QoS) context attributes and MobiShare supports string context matching. COSS supports only a limited set of predefined semantic attributes which can be classified into Boolean 'present' or 'not present'. However, this Boolean approach and a lack of reasoning support, provides no real advantage over string based mechanisms.

Table 1. Comparison of related architectures.

Architecture	Service Description and Matching	Context-aware Discovery
Jini	Interface/attribute string comparison	None
UPnP	Service name string comparison	None, but supports state change events
SLP	Service type/attribute string comparison	None
Salutation	Service type/attribute string comparison	None
Anamika	Ontological service type only	None
DReggie	DAML predefined ontology	None
Konark	Service type or keyword string comparison	None, but supports state change events
SSDM	String comparison and auctioning	Limited to QoS
MobiShare	Ontological service type	Context string comparison
COSS	Ontological service type, outputs and inputs	Boolean ontological attributes
Jini Context	See Jini	Limited to QoS
LUDS	String based comparison	State context support

Existing pervasive discovery architectures lack the rich ontological representation required to express the functionalities and capabilities of services, that separate parties can agree upon and understand [15]. They also fail to support the broad range of context information. This results in discovery of services which are less relevant to the mobile user's current situation or device constraints.

Semantic languages are still in an early stage of development and evaluation. This and their inherent complexity have led to their slow uptake despite the promised benefits. Inclusion of semantic languages into heavyweight reasoners such as Prolog [17], Clips [18] and Jess [19] has been slow. Furthermore, the memory intensive nature of these reasoners also presents challenges for incorporating reasoning into pervasive environments. Lack of semantically represented context has hampered the effectiveness of context-aware discovery. Challenges include addressing how to represent context semantically and whether OWL is sufficiently rich.

3 Improving Recall and Precision – An Illustrative Argument

In this section we demonstrate how the integration of semantics and context will improve the relevance of services, discovered in pervasive environments. We use the metrics of precision and recall [20] to highlight this improvement.

3.1 Printer Service Scenario

Bob wishes to print a document from his PDA, but does not know where to find a printer on his university campus. Bob issues the service request listed in Table 2. It contains two kinds of attributes: static and dynamic. Static attributes are those in which the value changes rarely, while in dynamic attributes the value changes regularly. Bob's static attributes indicate he wants the cheapest, non-colour printer. His dynamic attributes indicate he wants the closest printer which is available to students, ready (with paper and no paper jam) and with the shortest print queue. Each attribute is also specified as either a vital or rank attribute. Vital indicates a service must match the attribute in order to be included in the list of discovered services, while rank indicates the attribute will assist in determining a ranking index for the service and is associated with a weighting of importance to the requester. Any requested attribute which is unspecified by the service description, is not ruled out, but given lowest ranking for that attribute.

Table 3 lists Bob's current context. Table 4 presents the services currently on offer, listing service type and other attribute values for the service. The last column indicates whether the service is relevant to Bob. Since Bob specifies colour, available and ready as vital attributes, only three services are actually relevant to him.

Table 2. Bob's service request summary.

Attribute Name	Value	Vital/Rank	Weighting
Service type	Printer	Vital	
Colour	No	Vital	
Price	Cheap	Rank	0.3
Available	Yes	Vital	
Ready (has paper, no jam, etc)	Yes	Vital	
Location	Close	Rank	1
Paper Jams	Low	Rank	1
Print Queue	Short	Rank	0.7

Table 3. Bob's current context.

Name	Value
Location	Outside Building A

Table 4. Services on offer.

#	Type	Col-our	Price	Avail-able	Ready	Location	Paper Jams	Print Queue	Rel-evant
1	Laser Printer	Yes	50c	Yes	Yes	Building A	0	5	No
2	Laser Printer	No	15c	No	No	Building B	0	0	No
3	Laser Printer	Neg-ative	15c	Affir-mative	Affir-mative	Building C	0.3	0	Yes
4	Laser Copier	No	10c	Yes	Yes	Building G	0.01	5	Yes
5	Ink Jet	Yes	30c	No	No	Building A	0.3	0	No
6	Ink Jet	No	15c	Yes	Yes	Building F	0.2	3	Yes
7	Printer	No	9c	No	Yes	Building L	0.05	0	No
8	Printer Magazine								No
9	Printing Agency					Closed			No

3.2 Impact of Semantics and Context on Recall and Precision

Bernstein and Klein [20] define two quality metrics to assess the quality of services returned in a service list, with respect to relevance to the requester. These are recall and precision. Recall measures how well a discovery architecture retrieves all the relevant services; and precision, how well the architecture retrieves only the relevant services. We define a service to be relevant if it is judged so by the requesting user.

In order to determine recall and precision: Let *x* be the number of relevant services returned to the requester. Let *n* be the total number of relevant services available. Let *N* be the total number of services returned.

$$\text{Recall} = x / n \quad (1)$$

$$\text{Precision} = x / N \quad (2)$$

Supporting semantics improves both recall and precision, while supporting context-aware discovery improves only precision. This is because context-aware discovery involves 'ruling out' irrelevant services and does not 'rule in' services that have already been discounted. However, since semantics affects both measures, contextual attributes which are not described semantically would reduce both precision and recall, because services could be ruled out due to misinterpreted contextual representation. For instance, the 'printer ready' attribute could hold the attribute 'affirmative', but ruled out because this was not equal to 'yes'.

Current architectures match request attributes with service description attributes, based on exact string matching and do not support context attributes. Table 5 presents a ranked service list resulting from the discovery process using syntax based techniques, where services are matched on service type alone. Context attributes are also omitted since they are not supported. The vital attributes are used to determine which services will comprise the list of discovered services. In terms of service type, only those services which contain the word 'printer' would be discovered. Service 7 would be ranked first because it matches this keyword exactly. Service 8 and 9, which refer to a magazine called 'Printer' and a printing agency, respectively, are discovered, but are not relevant.

The recall ratio resulting from Table 5 is 1/3 or a decimal value of 0.33. The precision ratio is 1/6 or decimal of 0.17. These results highlight the limited effectiveness of such techniques because five irrelevant services were discovered and only one of three relevant services were discovered.

Table 5. Service list returned by current architectures.

#	Type	Colour	Price	Relevant
7	Printer	No	9c	No
1	Laser Printer	Yes	50c	No
2	Laser Printer	No	15c	No
3	Laser Printer	Negative	15c	Yes
8	Printer Magazine			No
9	Printing Agency			No

If static attributes 'colour' and 'price' were matched during discovery service 1 would be correctly ruled out, but service 3 would also be incorrectly ruled out because 'negative' does not match the requested 'no' colour value, even though it has the same meaning. No relevant services would be discovered. The ‘price’ attribute cannot be utilised by current architectures, because they cannot interpret the meaning of ‘cheap’ in Bob’s service request. No service has the value ‘cheap’ for ‘price’.

If current architectures were extended to support context attributes, the service list provided in Table 6 would be returned. Since ‘available=yes’ and ‘ready=yes’ are vital request attributes, services 7, 1, 2 were correctly removed from the discovered services list. However, service 3 was also pruned from the list because ‘negative’ and ‘affirmative’ were different symbols to ‘no’ and ‘yes’, respectively. Consequently, only services 8 and 9 were discovered. Both of these services are completely irrelevant to Bob, as they do not assist him in his goal of printing a digital document. Attributes such as ‘price’, ‘location’, ‘paper jams’ and ‘print queue’, were also interpreted incorrectly. For instance, Service 9 was ranked higher than service 8, because the requested ‘location=close’ matched service 9’s ‘location=closed’. This is an example of a homonym in which terms are syntactically similar, but semantically different. Service 9’s ‘closed’ value indicates that the agency is currently not trading for business through its store front. This results in a zero finding for both recall and precision.

Table 6. Service list returned by current architectures with context.

#	Type	Colour	Price	Available	Ready	Location	Paper Jams	Print Queue	Rele-vant
9	Printing Agency					Closed			No
8	Printer Magazine								No

We propose a semantic data representation be adopted for pervasive service discovery, to ensure that each request attribute is matched with service description attributes based on meaning rather than symbol. Context information must also be utilised and represented semantically. Table 7 presents the services discovered under this approach, when considering vital attributes. In his request, Bob indicated he sought a service of the type printer, which was matched as having the same meaning as ‘Laser Printer’ and ‘Ink Jet’. The requested colour attribute’s ‘no’ value was matched with ‘no’ and ‘negative’. For the ‘available’ and ‘ready’ attributes, ‘yes’ was matched with ‘yes’ and ‘affirmative’.

Table 7. Discovered service list returned by proposed model using vital attributes.

#	Type	Colour	Available	Ready
3	Laser Printer	Negative	Affirmative	Affirmative
6	Ink Jet	No	Yes	Yes
4	Laser Copier	No	Yes	Yes

Table 8 shows how the services were ranked to provide a ranked service list. Each rank attribute of each service, was ranked relative to the corresponding attribute of the other services in the discovered set. This provides a level of match for each attribute of a service, with the request. For instance, in the case of the ‘frequency of paper jams’ attribute, service 4 was ranked first because it had the fewest paper jams (frequency of 0.01) comparative to the other services discovered, followed by service 6 then 4. The rankings are then weighted according to the weighting of importance to the requester, specified in the service request, to provide an effective rank. For each service, the effective rankings for each attribute are added together to provide an overall rank index for that service. The service with the lowest overall rank index, is ranked first in the final list returned to the requester.

Using the measures of recall and precision, it can be seen that bringing together 'semantics' and 'context' will dramatically improve the effectiveness of the service discovery process, returning only the most relevant services to the requester. The resulting recall ratio increases to 1, based on the service list presented in Table 7 and Table 8, from 0.33 under syntax matching (Table 5). The precision ratio result increases to 1 from 0.17.

Table 8. Service list ranking achieved by proposed model.

#	Price	Effective Rank	Location	Effective Rank	Paper Jams	Effective Rank	Print Queue	Effective Rank	Overall Rank
3	15c	*2*0.3=0.6*	Building C	*1*1= 1*	0.3	*3*1= 3*	0	*1*0.7=0*	0.6+1+3 +0.7=5.3
6	15c	*2*0.3=0.6*	Building F	*2*1= 2*	0.2	*2*1= 2*	3	*2*0.7=1.4*	0.6+2+2 +1.4=6
4	10c	*1*0.3=0.3*	Building G	*3*1= 3*	0.01	*1*1= 1*	5	*3*0.7=2.1*	0.3+3+1 +2.1=6.4

Recall has improved because all the relevant services were discovered and precision improved because only the relevant services were discovered. This shows there is a need for a pervasive service discovery model that facilitates service matching using semantics and context. We now present our preliminary conceptual model for pervasive service discovery.

4 Preliminary Conceptual Model

In this section we present our preliminary conceptual model, to realise our proposed approach for bringing together 'semantics' and 'context'.

4.1 Service Discovery

In our preliminary conceptual model we have several modules which support and utilise semantically represented information and realise dynamic context attributes (Fig 1). Existing discovery protocols are either end-to-end or involve a third party. Dabrowski and Mills [21] term this third-party a service cache manager. Our model may reside on a service cache manager or at the requester or provider.

Service providers advertise their services in the form of OWL-S service profiles, to the Advertiser Module, which stores these profiles into the Service Description Repository. Service profiles contain the functional and non-functional attributes, which may include: inputs, outputs, preconditions, constraints, requirements and other service attributes. Each of these attributes can be related to web-based ontologies to give them semantic meaning.

Service requesters submit requests in the form of OWL-S service profiles, to the Requester Module, which instructs the Discovery Manager to match the request with services contained in the Service Description Repository. The requester's current context can be transparently obtained from the User and Device Repositories without the user's explicit knowledge. The requester will be provided with a ranked list containing services which exactly or approximately match the request.

We discuss the functionality of the modules in Fig 1, in the next sections.

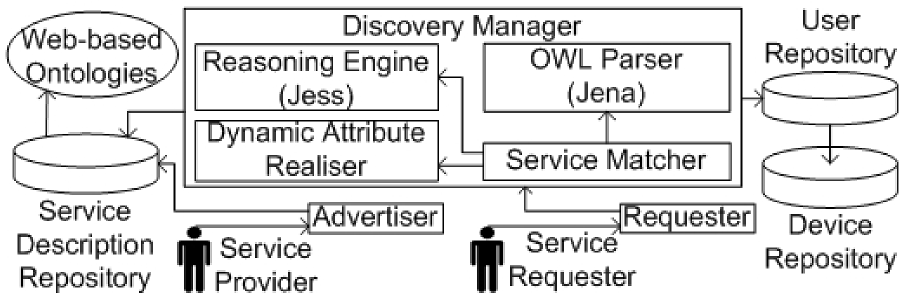

Fig. 1. Conceptual model for semantic context-aware discovery.

4.2 Semantic Management

In our architecture, static attributes comprise name and value pairs represented semantically as ontological concepts, such that they can be reasoned about. When the Discovery Manager receives a service request, the OWL Parser module, implemented in Jena [22], is enacted to load and verify OWL-S service profiles and associated ontologies and represent these as objects in our architecture. The Service Matcher will obtain a set of candidate services from the Service Repository and transform these into objects using the OWL Parser. The Service Matcher enacts the Reasoning Engine, implemented in Jess [19], to perform matching between the service request and individual services. It loads the parsed OWL-S service profiles, related web-based ontologies, OWL relationship rules (for example, inverse, equivalence, disjoint, etc.) and rules that will perform matching. The reasoner will rank each service for its match to each functional and non-functional attribute defined in the request, as in section 3.

4.3 Context Management

Dynamic attributes hold context information and are represented in the same way as static attributes except their value is not determined until needed. The Service Matcher utilises the Dynamic Attribute Realiser, to obtain their value, before matching takes place. There may be two sides to a dynamic attribute to be obtained: the requester's context constraints or the service's context requirements. Both sides will not always be dynamic attributes; one side could be static. The requester's dynamic attributes are either explicitly contained within the service request or implicitly provided by one of two repositories:

- **User Repository**: Details relating to the human user of the requesting device will be stored in this repository. These may include user preferences explicitly provided by the user when he or she used the system for the first time or data implicitly collected by the architecture. Implicit data may include historical usage patterns and user behaviour such as services previously invoked by the user. Implicit context may be collected using a third-party architecture which has yet to be decided on and is the subject of continued investigation.

- **Device Repository**: Each user will be associated with one or more devices. Each device has its own set of specifications, such as processing capability, screen size, input method, etc. This information will be collected from the device when it registers with the system for the first time.

The service's dynamic attributes will be specified in the service's description located in the Service Description Repository. A service can be associated with two main kinds of dynamic attributes: (1) dynamic attributes which can be determined locally by the discovery manager and (2) those which must be remotely determined by the service provider. An example for the first type is quality of service (number of network hops or delay). The second type, falls into two separate subcategories. Attributes which are constantly changing (for example service state such as a locked door) and those which change less often (for example server load or print queue). The discovery manager must remotely query the service provider for the current value of constantly changing attributes or is notified by the provider when the value changes on a less frequently changing attribute. The provider will specify each attribute's category, in the service description. Once the dynamic attributes have been realised, they are returned to the service matcher for matching, as with static attributes.

We are currently implementing this model in Java. We are using the Mindswap OWL-S API [23] and Jena [22] for the OWL representation and OWL Parser module in our model. We are using the OWLJessKB [24] API and Jess [19] to implement the Reasoning Engine.

5 Conclusion and Future Work

Our paper demonstrates that bringing together 'semantics' and 'context' during pervasive service discovery will improve the relevance of services to the requester. This is paramount due to the constraints (time and resource) and dynamic/volatile nature of pervasive environments. Thus, the main contribution of our paper is our analytical illustration of the impact of semantics and context on precision and recall.

We also propose a preliminary conceptual model for achieving this and we are implementing this model to create a prototype. We are working to address the significant challenges involved in implementing this model on resource constrained small devices. In future work we seek to expand the model's ability to discover the more relevant services, by not only utilising semantics and context, but also other mechanisms for service filtering such as user preferences, historical data and others.

References

1. Broens T., Context-aware, Ontology based, Semantic Service Discovery [Master], Enschede, The Netherlands, University of Twente, 2004.

2. O'Sullivan J., Edmond D. and Hofstede A. H. M. t., Service Description: A Survey of the General Nature of Services, April, 2002.
3. OWL Services (OWL-S) 1.1, http://www.daml.org/services/owl-s/1.1/.
4. Doulkeridis C., Loutas N. and Vazirgiannis M., A System Architecture for Context-Aware Service Discovery, 2005.
5. Doulkeridis C. and Vazirgiannis M., Querying and Updating a Context-Aware Service Directory in Mobile Environments, In Proc. 4th VLDB Workshop on Technologies on E-Services (TES'03), 2003.
6. Arnold K., O'Sullivan B., Scheifler R. W., Waldo J. and Woolrath A., The Jini Specification, Addison-Wesley, 1999.
7. Universal Plug and Play (UPnP), http://www.upnp.org.
8. Guttman E., Perkins C. E., Veizades J. and Day M., Service Location Protocol, Version 2, IETF, RFC 2608, June, 1999.
9. Guttman E., Service Location Protocol : Automatic Discovery of IP Network Services, IEEE Internet Computing, 1999, 3(4), 71 - 80.
10. Miller B. A. and Pascoe R. A., Salutation Service Discovery in Pervasive Computing Environments, IBM Pervasive Computing White Paper, February, 2000.
11. Lee C., Helal A., Desai N., Verma V. and Arslan B., Konark: A System and Protocols for Device Independent, Peer-to-Peer Discovery and Delivery of Mobile Services, IEEE Transactions on Systems, Man and Cybernetics, November, 2003, 33(6).
12. Issarny V. and Sailhan F., Scalable Service Discovery for MANET, In Proc. Third Annual IEEE International Conference on Pervasive Computing and Communications (PerCom), 8 - 12 March, Kauai Island, Hawaii, 2005.
13. Chakraborty D., Joshi A., Yesha Y. and Finin T., Towards Distributed Service Discovery in Pervasive Computing Environments, IEEE Transactions on Mobile Computing, 15 July, 2004.
14. Norvag K., Polyzos G. C., Valavanis E., Vazirgianis M. and Ververidis C., MobiShare: Sharing Context-Dependent Data and Services among Mobile Devices, In Proc. IEEE/WIC International Conference on Web Intelligence (WI'03), October, 2003.
15. Chakraborty D., Perich F., Avancha S. and Joshi A., DReggie: Semantic Service Discovery for M-Commerce Applications, In Proc. Workshop on Reliable and Secure Applications in Mobile Environment, In Conjunction with 20th Symposium on Reliable Distributed Systems (SRDS), 12 October, 2001.
16. Catterall E., Davies N., Friday A., Pink S. and Wallbank N., Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments, Wireless Networks, 2004, 10, 631-641.
17. SWI-Prolog, http://www.swi-prolog.org/.
18. CLIPS: A Tool for Building Expert Systems, http://www.ghg.net/clips/CLIPS.html.
19. Friedman-Hil E., Jess: Java Expert System Shell, the Rule Engine for the Java Platform, http://herzberg.ca.sandia.gov/jess/, updated: 2 Nov 2005, accessed: 6 Nov 2005, Sandia National Laboratories.
20. Bernstein A. and Klein M., Towards High-Precision Service Retrieval, In Proc. International Semantic Web Conference (ISWC 2002), 9 - 12 June, 2002, p. 84 - 101.
21. Dabrowski C. and Mills K., Analyzing Properties and Behaviour of Service Discovery Protocols using an Architecture-based Approach, In Proc. Working Conference on Complex and Dynamic Systems Architecture, 2001.
22. Jena - HP Semantic Framework, http://www.hpl.hp.com/semweb/, Hewlett Packard.
23. Mindswap: OWL-S API, http://www.mindswap.org/2004/owl-s/, University of Maryland Institute for Advanced Computer Studies.
24. OWLJessKB: A Semantic Web Reasoning Tool, http://edge.cs.drexel.edu/assemblies/software/owljesskb/, Geometric and D. U. Intelligent Computing Laboratory.