Context-aware, ontology-based, service discovery

Tom Broens[1,2], Stanislav Pokraev[1], Marten van Sinderen[2], Johan Koolwaaij[1], Patricia Dockhorn Costa[2]

[1] Telematica Instituut, P.O. Box 589,
7500 AN Enschede, The Netherlands
{Tom.Broens, Stanislav.Pokraev, Johan.Koolwaaij}@telin.nl
http://www.telin.nl/
[2] Center for Telematics and Information Technology,
University of Twente, P.O. Box 217
7500 AE Enschede, The Netherlands
{t.h.f.broens, m.j.vansinderen, p.dockhorncosta}@ewi.utwente.nl
http://www.ctit.utwente.nl/

Abstract. Service discovery is a process of locating, or discovering, one or more documents, that describe a particular service. Most of the current service discovery approaches perform syntactic matching, that is, they retrieve services descriptions that contain particular keywords from the user's query. This often leads to poor discovery results, because the keywords in the query can be semantically similar but syntactically different, or syntactically similar but semantically different from the terms in a service description. Another drawback of the existing service discovery mechanisms is that the query-service matching score is calculated taking into account only the keywords from the user's query and the terms in the service descriptions. Thus, regardless of the context of the service user and the context of the services providers, the same list of results is returned in response to a particular query. This paper presents a novel approach for service discovery that uses ontologies to capture the semantics of the user's query, of the services and of the contextual information that is considered relevant in the matching process.

1 Introduction

Ambient intelligence aims at enriching users' lives by providing ubiquitous, transparent and intelligent electronic services [1]. These services are diverse and distributed in the user's environment.

A key feature of ambient intelligence is transparency on service provisioning. The process of discovering and invoking relevant services should be hidden from the users' point of view. In order to realize this scenario, we need mechanisms to provide smart service discovery based on the current situation of the user (e.g., user's location, his interest, user's environment characteristics, etc). We define the user's current situation as context [9]. Contextual information of the user is therefore an essential aspect to accomplish transparency in the service discovery process within the ambient intelligence scenario.

Most of the existing service discovery mechanisms retrieve services descriptions that contain particular keywords from the user's query. In the majority of the cases this leads to low recall[1] and low precision[2] of the retrieved results. The reason for the first is that query keywords might be semantically similar but syntactically different from the terms in service descriptions, e.g. 'buy' and 'purchase' (synonyms). The reason for the second is that the query keywords might be syntactically equivalent but semantically different from the terms in the service description, e.g. 'order' in the sense of proper arrangement and 'order' in the sense of a commercial document used to request supply of something (homonyms). Another problem with keyword-based service discovery approaches is that they cannot completely capture the semantics of user's query because they do not consider the relations between the keywords. One possible solution for this problem is to use ontology-based retrieval. In this approach, ontologies are used for classification of the services based on their properties. This enables retrieval based on service types rather than keywords.

Another drawback of the existing service discovery approaches is that the query-service matching score is calculated taking into account only the keywords from the user's query and the terms in the service descriptions. Thus, regardless of the context of the user and the context of the service providers, the same list of results is returned in response to a query. By definition, context is a situation of an entity (person, place or object) that is relevant to the interaction between a user and an application [9]. Therefore, considering the context in the query-service matching process can improve the quality of the retrieved results. However, contextual information is highly interrelated and has many alternative representations [27] that makes it difficult to interpret and use. One possible solution is again to use ontologies to specify the interrelations among context entities and ensure common, unambiguous representation of these entities.

This paper presents a novel approach for service discovery that uses ontologies to capture the semantics of the user's query, of the services and of the contextual information that is considered relevant in the matching process. The paper is based on a master thesis [6] that can be used as further reading.

The paper is structured as follows: section 2 presents the existing service discovery approaches and their major drawbacks. Section 3 presents our service discovery approach. Section 4 discusses the implementation and evaluation of the proposed approach and section 5 summarizes our contributions.

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

[2] 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

2 Existing service discovery approaches

2.1 Traditional service discovery

CORBA [24] proposed one of the first service discovery approaches. It specifies naming [23] and trading services [22] used to discover objects on a network. The naming service is keyword-based whereas the trading service supports discovery based on the service types. *UDDI* [29] is the most used service discovery approach for web services [3]. The core of the UDDI architecture is a central business registry that functions as a naming and directory service. Services in this registry are described from three different perspectives, comparable to the white, yellow and green pages of the telephone dictionary. Furthermore, service descriptions consist of tModels that classify the business or web service using standard or user-defined taxonomies. *OSGi* [25] proposes an open service platform for the delivery of applications and services to all types of networked devices. Service discovery is performed by querying the name or the type of a service. *OSGa* [14] focuses on integration of grid computing paradigms with web services technologies. The service is advertises by its service information (i.e. name, type) in the registry. By retrieving this service information, the user can discover services.

Klein [19] discusses several categories of service discovery technologies and their limitations for the quality of the service discovery result. According to Klein's categories, the traditional service discovery approaches are either keywords-based or table-based and they don't take into account the contextual information. As discussed in the introduction this leads to low quality of the retrieved results.

2.2 Context-aware service discovery

This section presents the existing approaches that consider the contextual information in the service discovery process. It also discusses the problems of using contextual information in those approaches.

The *Cooltown* [15] project allows users to discover services that are in the user's vicinity. In this approach the location of the user and the service is used to derive that the user is in the service area. This way, services that are close to the user are returned by the service discovery mechanism. *The context toolkit* [8] is a development toolkit that provides functionality to discover services using contextual information. It allows for describing services by means of white and yellow pages that include contextual information. *The platform for adaptive applications* [10] proposes architecture for applications that adapt their behavior according to the context of the user. The platform enables discovery of context providers by the type of context they advertise. This contextual information is used to adapt the application behavior. The *CB-Sec* project [20] provides functionality to discover services that are in the vicinity of the user. This approach takes into account the user and service capabilities in the service discovery process.

The contextual information is highly interrelated and has many alternative representations [27]. This makes it difficult to interpret and use. Context providers and context consumers (e.g. service providers or requestor) may have different understandings of the same contextual information. This leads to misinterpretation of the information, which in turn leads to misunderstanding of the user goal and therefore poor discovery results.

2.3 Ontology-based service discovery

As we said earlier, shared understanding on the concepts, used to describe services and contextual information, is crucial to ensure high quality service discovery results. The required, shared understanding can be provided by the use of ontologies [11]. There are several approaches that use ontologies in the service discovery process. However, none of them considers the use of contextual information in the service discovery process.

OWL-S [29] is an OWL [31] service ontology that can be used to semantically describe services. It allows specification of services in terms of their *inputs*, *outputs*, conditions, that have to hold true before the service execution (called *preconditions* in OWL-S terms), and post-conditions, that represent the state of the environment after the service execution (called *effect* in OWL-S terms). *COBRA* [7] divides the world into different application domains. Each domain is specified by its own ontology that provides shared concepts and relations for service discovery. *OntoMat* [2] uses ontologies to map the concepts used by the service requestor to the concepts used by the service provider. This way, those concepts can be compared and reasoned about. *CBSDP* [18] is a service discovery protocol for ad hoc networks. CBSDP uses ontologies to interpret the data exchanged during service execution.

3 Our approach

We argue that the use of contextual information in the service discovery process increases the recall and precision of the retrieved results. On the one hand, the contextual information makes the user's query more information-rich and thereby provides means for higher precision of the retrieved results, that is, the context helps to capture better the user's goal. 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. However, as discussed in 2.2., contextual information is very complex and has many alternative representations. Therefore, we propose to use ontologies to model such information. The use of ontologies for describing users' queries, service properties and contextual information is advantageous. First, ontologies provide a *vocabulary* for modeling knowledge in a restricted domain. They are built by reaching a consensus within a community of interest and thus are a key enabler for seamless knowledge interchange. Second, ontology languages are usually grounded with *formal semantics* such as model theory or description logic. This in turn enables *unambiguous definitions of compound concepts*. Based on these definitions it is possible to *infer* new implicit information from present (explicit) information. Finally, the *common vocabulary* and precise mathematical specification of semantics open the way to *automatic information processing* since the information is not only understood by humans but also by machines.

3.1 Positioning

Figure 1 shows the position of our approach with respect to the existing service retrieval approaches identified by Klein in [19].

We position our approach in the space between the concept-based approach and deductive retrieval approach. The deductive approach offers higher recall and precision, however, modeling service functionality by the means of formal logic is sometimes an extremely difficult task. Another disadvantage of the deductive approach is that the search process is usually very slow due to the high computation complexity of the proof process.

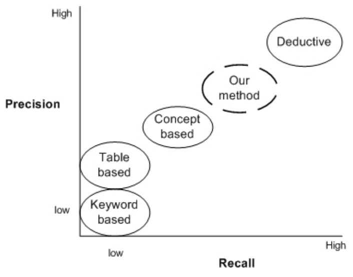


Fig. 1. Positioning of our approach

3.2 Overview

In our approach, we distinguish several high-level components (fig. 2). The inputs of our matching component are: the user's query (i.e. the service request), a set of advertised services (i.e. service descriptions), a set of context providers, and the ontologies, used by the user, service and context providers.

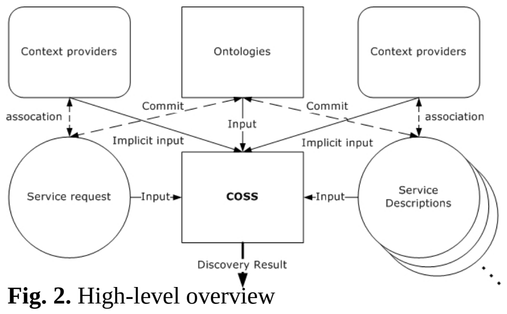


Fig. 2. High-level overview

In our approach service users, service providers and context providers achieve a shared understanding by using ontologies to which they all commit. Users and service providers have associated context providers that can deliver different types of contextual information, for instance, user location or weather conditions in a certain service area. To enable unambiguous, knowledge interchange, our approach uses domain-specific ontologies. In such ontologies, concepts from a particular domain and relations among them are precisely specified. This enables reasoning on the user queries, service descriptions and associated contextual information. For instance, consider a shop that advertises: sale of 'music products'. If a user specifies that he wants to buy a 'music CD', the query and the service description do not match syntactically. If we employ domain-specific ontologies to derive that 'music CD' is a 'music product', we can conclude that the query and the service description match semantically.

We distinguish four different service properties that are handled differently by our matching algorithm:

- *Service type*: Refers to an entry in some ontology or taxonomy of services. Example of such an ontology is the UNSPSC[3] classification system.
- *Outputs*: Refers to a concept from a domain-specific ontology that specifies the value that this particular service delivers to its environment (e.g. music products, traffic information, etc.)
- *Inputs*: Refers to a concept from a domain-specific ontology that specifies the sacrifice a user is ready to make in order to receive the value delivered by a service (e.g. money, effort to fill in a questionnaire, etc.)
- *Contextual attribute*: Represent the contextual information derived from the user (e.g. user location) and service providers (e.g. service location).

3.2 Service grounding

To be able to invoke a service after its discovery, in our approach we use a WSDL grounding mechanism. WSDL [32] defines *services* as collections of network *endpoints*. The abstract definition of an endpoint, called *interface*, is separated from its concrete network deployment, *protocol* and *data encoding* through reusable *bindings*. *Interfaces* are abstract collections of *operations* that contain input and/or output *messages* which consist of *message parts*. Fig. 3 presents the mapping between our service model and the WSDL metamodel. In our service model each service has a service type. This service type is mapped to a WSDL interface. The service itself maps to an operation in this interface (e.g. SellMusicCD). The inputs and outputs of the service map to messages in WSDL whereas concepts map to message parts. The following example outlines our grounding mechanism.

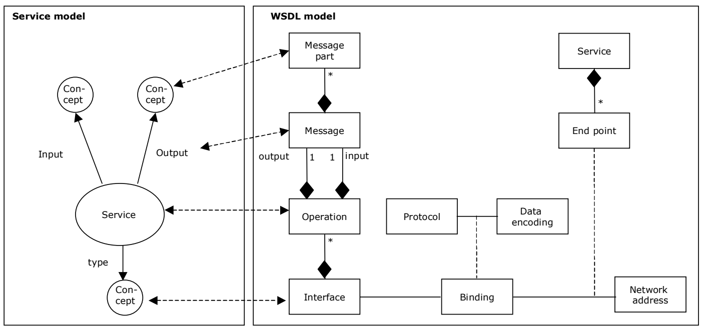


Fig. 3. Service model and mapping to the WSDL metamodel

```
...
<operation name="SellMusicCD">
  <input message="credit_card"/>
  <output message="CD" />
```

[3] http://www.un-spsc.org

```
</operation>
<message name="credit_card">
  <part name="type" payontology:output="payontology:#CreditCardType" />
  <part name="card" payontology:output=" payontology:#Card"/>
  <part name="expire" payontology:output=" payontology:#ExpireDate" />
</message>
...
```

3.3 Matching algorithm

Our approach matches a user query with a set of available service descriptions. The result is a set of service descriptions that semantically match the user query. To rate the matches we defined a quality measure called *matching degree*.

Matching degree

Consider a user request R and a service description S. To rate how relevant particular match between R and S is, we use the number of service properties (i.e. type, inputs, outputs and contextual attributes) from the request that are not present in S. Based on those missing properties we classify the match in five different categories, defined by Li [21] (fig. 4).

The first category indicates an *exact match*. The request has the same properties as the service description, i.e. there are no missing properties. This is the best possible match. The second category is called *plug-in match*, that represents the second best match. It indicates that the service is capable of more than the requestor wants. The third and fourth category, called *subsume match* and *intersection match*, respectively, indicate that the service can only partially provide what the user wants, i.e. the number of missing properties is bigger than zero. The fifth match category indicates a *disjoint match*, i.e. the request and the service do not share any properties.

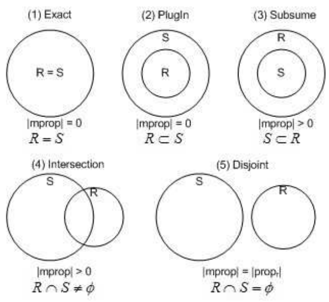


Fig. 4. Match categories

Our approach uses this initial classification to further classify matches in three types of matches that are useful for the user:

- *Precise match*: Exact and Plug-in matches. The service is capable of providing the requested functionality or more.
- *Approximate match*: Subsume and intersection matches. The service is capable of providing part of the requested functionality.
- *Mismatch*: Disjoint match. The service is not capable of providing the requested functionality and will not be returned to the user.

Algorithm

The goal of the matching algorithm is to classify the available set of services using the service request into the three previously defined matching types. This is done in four steps (fig. 5).

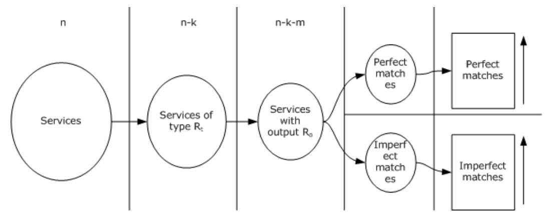


Fig. 5. Matching algorithm

The starting point of the matching process is a set of all service (S) available to the matchmaker (e.g. n). The first step will filter out those services that are not of the desired service type provided in the user request (R). This results in a smaller set of services (e.g. n-k) with service type R_t. The second step will filter out all service descriptions that do not have the desired service output. Again, this results in a smaller set of services (e.g. n-k-m) that can provide the requested output R_o. The services of this set are then queried for the inputs (s_i) they require. If the required inputs are provided by the user or can be provided by the context providers (e.g. when the service needs as input the user location that is not provided by the user but by the user location context provider) the match is classified as perfect. Else the match is classified as imperfect. The final step orders the two sets using the contextual attributes (discussed in the next section). All phases are represented in the following matchmaking algorithm.

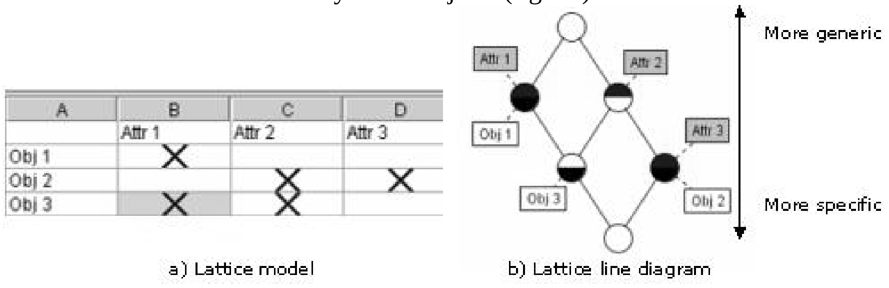
```
Matching(R, S) {
    S’ = query_Registry($R_t$, S)
    S’’ = query_Registry($R_o$, S’)
    forall s in S’’ do {
             $s_i$ = query_Inputs(s)
             if provided($s_i$,$R_i$) then {
                   Precise.append(s)
             }
             else {
                   if query_ContextProviders(userID,
                   missing_Inputs($s_i$, $R_i$)) then
                 {
                      Precise.append(s)
                 }
                 else {
                      Approximate.append(s)
                 }
             }
    }
    P = order_with_ContextualAttributes(Precise)
    A = order_with_ContextualAttributes(Approximate)

    return result(P, A)
}
```

Contextual attributes model

Users can define some preferences about certain properties of a service they want to discover. This can for instance be the preference *nearby* that defines that the user wants to retrieve a service close to him. We call these service properties/user preferences “contextual attributes”. The contextual attributes are defined in a simple rule: Attribute –definition-> Statement. The statement defines the meaning of the attribute (e.g. nearby –definition-> distance *(userposition, serverposition)* < maxdistance). These contextual attributes are used to order the sets of returned matches.

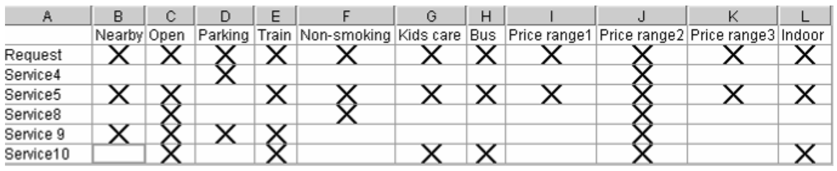
We use a clustering mechanism to rate services based on the preferences they have. For that purpose, we use concept lattices [13]. 'Concept lattices' is a mechanism used in formal concept analysis. It can be used to study how objects can be hierarchically grouped together according to their common attributes. The starting point is a concept model which consists of a triple (G,M, I). G is set of *objects*, M is a set of *attributes* and I is a *binary relation* between them ($I \subseteq GxM$). A common used representation of this model is a cross table (fig. 6a). Each object is one row in the table while the attributes are the columns. The binary relation is presented by a cross at the intersection of a row and a column. The lattice table is the basis for a lattice line diagram that visualizes the attribute communality of the objects (fig. 6b).



A	B	C	D
	Attr 1	Attr 2	Attr 3
Obj 1	X		
Obj 2		X	X
Obj 3	X	X	

Fig. 6. Concept lattices

This is a hierarchical diagram which presents the most generic objects at the top while getting down in the diagram the objects get more specific (i.e. have more attributes). A node in the diagram is called *concept* and can contain objects that share the same attributes. Such a concept shares the *attributes* from its parents in the diagram. The top node is a set of objects that contains no attributes. One level down object 1 is encapsulated by a concept that contains attribute 1. Again one level down we see that object 3 has a relation with the concept containing attribute 1 and with a concept containing attribute 2. Therefore, object 3 has attribute 1 and attribute 2 and shares attribute 1 with object 1. Object 2 has attribute 3 and shares attribute 2 with object 3. The bottom node contains objects that have all attributes (in this case empty). This model is analogues to our contextual attribute model, where a service (object) has some contextual attributes (attributes) (fig. 7).



A	B	C	D	E	F	G	H	I	J	K	L
	Nearby	Open	Parking	Train	Non-smoking	Kids care	Bus	Price range1	Price range2	Price range3	Indoor
Request	X	X	X	X	X	X	X	X	X	X	X
Service4			X						X		
Service5	X	X		X	X	X	X	X	X	X	X
Service8		X			X				X		
Service 9	X	X	X	X					X		
Service10		X		X		X	X		X		X

Fig. 7. Lattice cross table

The request and all retrieved services descriptions are added to this table as objects (rows). The preferences are evaluated for the services (cross) and added as attributes (columns). From this table a lattice line diagram is calculated (fig. 8).

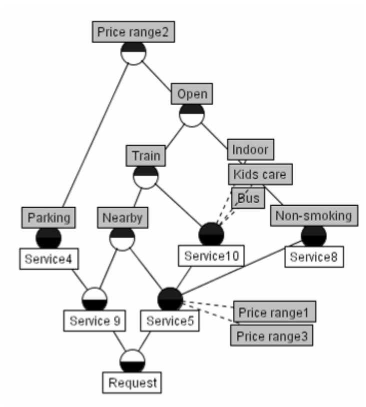


Fig. 8. Lattice line diagram

This diagram should be read from the top to the bottom. A child node shares the attributes of its parents (e.g. service 5, service 9 and request all have attributes nearby, train, open, price range2 etc). So, by reasoning on the position of services related to the position of the request an ordering of services can be made. Services positioned higher in the diagram than the request miss preferences. The higher the services are positioned the more preferences they miss the lower in the resulting list they are ordered.

4 Implementation and evaluation

Our approach was implemented as part of an experimental platform [12]. The platform provides the environment for mobile context-aware application to use third party content services (i.e. web services). The platform is implemented using Java technology. Parlay X [26] is used to interact with 3G network services while the AXIS framework [4] is used to interact with the third party content services. The client side is implemented using Personal Java and runs on a variety of embedded devices (e.g. smartphone, PDA).

Our approach is embedded in the matchmaker component of the experimental platform. Service advertisements are stored in MySQL databases as persistent Jena [17] models, and retrieved by executing RDQL [16] statements. The approach is implemented modular by encapsulating it in webservices. Therefore, the approach is not solely suitable for handling explicit requests by the user, but it is also able to deal with implicit requests, for instance, by an ambient intelligence environment.

We evaluated the approach using the implemented prototype. One of the evaluations issued queries using the prototype. Recall and precision rates where calculated and compared to recall and precision rates when using keyword based mechanisms. As an example, a query containing homonyms showed a gain of recall and precision of more than fifty percent. Further reading on the evaluation can be done in the master thesis [6].

5 Conclusion

In this paper we discuss the shortcomings of existing service discovery approaches and propose a novel approach [6] to overcome some of them. Our approach[4] uses the available contextual information about a particular user or service provider (e.g. user location or service opening times). In addition, it uses ontologies to semantically express user queries, service descriptions and the contextual information.

The use of contextual information in our approach resulted in higher quality of the retrieved results. On the one hand, the contextual information makes the user's query more information-rich (e.g. by adding extra information about the user's preferences) and thereby increases the precision of the retrieved results. On the other hand, the contextual information serves as an implicit input to a service that is not explicitly provided by the user. This allows our matching algorithm to select services that would be filtered out otherwise, which leads to higher recall of the retrieved results.

Besides the use of contextual information, we showed that use of ontologies in the context-aware, service discovery has many advantages. First, ontologies provide a shared vocabulary for specification of user queries, of service descriptions and of contextual information. This provides a basis for matching of meaningful user queries and meaningful service descriptions rather than just syntactic textual descriptions. Second, we used OWL, which is grounded with formal semantics of the Description Logic [5]. This allowed us to define unambiguously compound concepts and to reason about them.

Finally, the use of concept lattices for clustering services with similar attributes provided a convenient way to order services by their relevancy for the user. However, the designed mechanism is just a first step on using concept lattices in service discovery. Our future work includes a broader inspection of the use of concept lattices in service discovery.

References

1. Aarts, E., Eggen, B., Ambient Intelligence in Homelab (2002), Philips Research
2. Agarwal, S., Handschuh1, S., Staab, S., Surfing the Service Web (2003), ISCW'03.
3. Alonso, G, Casati, F, Kuno, H, Machiraju, V, Web services: Concepts, Architectures and Applications (2003), ISBN 3-540-44008-9
4. Apache webservices projects, AXIS (2004), http://ws.apache.org/axis/.
5. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., The Description Logic Handbook (2003), ISBN 0-521-78176-0.
6. Broens, T.H.F., Context-aware, Ontology based, Semantic Service Discovery (2004), Master thesis, University of Twente, the Netherlands
7. Chen, H., An Intelligent Broker Architecture for Context-Aware Systems (2003), PhD thesis, University of Maryland Baltimore County.

[4] This work is part of the Freeband AWARENESS project (http://awareness.freeband.nl). Freeband is sponsored by the Dutch government under contract BSIK 03025.

8. Dey, A., The context toolkit (2004), http://www.cs.berkeley.edu/~dey/context.html.
9. Dey, D., Providing Architectural Support for Context-Aware applications (2000), PhD thesis, Georgia Institute of Technology.
10. Efstratiou, C., Cheverst, K., Davies, N., Friday, A., An architecture for the Effective Support of Adaptive Context-Aware Applications (2001), Mobile Data Management 2001, p.15-26.
11. Fensel, D., Ontologies: A silver bullet for Knowledge Management and Electronic-Commerce (2000), ISBN 3-540-41602-1.
12. Freeband, WASP project (2003), http://www.freeband.nl/projecten /wasp/ENindex.html.
13. Ganter, B., Stumme, G., Formal concept analysis: Methods and Applications in Computer Science, TU Dresden, http://www.aifb.uni-karlsruhe.de/WBS/gst/FBA03.shtml.
14. Globus, Open Service Grid Architecture (2004), http://www.globus.org/ogsa/.
15. Hewlet Packard, Cooltown project, http://www.cooltown.com/cooltown/index.asp (2004) .
16. Jena community, A Programmer's Introduction to RDQL (2002), http://jena.sourceforge.net/tutorial/RDQL/index.html.
17. Jena community, Jena – A Semantic Web Framework for Java (2004), http://jena.sourceforge.net.
18. Khedr, M., Enhancing service discovery with Context Information (2002), ITS'02, Brazil.
19. Klein, M., Bernstein, A., Towards High-Precision Service Retrieval (2004), IEEE Internet Computing, January, p. 30-36.
20. Kouadri Mostefaoui, S., Tafat-Bouzid, A., Hirsbrunner, B., Using Context Information for Service Discovery and Composition (2003), Proc. of the 5th Conf. on information integration and web-based applications and services, Jakarta, p. 129-138.
21. Li, L., Horrocks, I., A software framework for matchmaking based on semantic web technology (2003), In Proc. of the 12th Int. World Wide Web Conference, p. 331-339.
22. Object Management Group, Catalog of Corba Facilities specifications (2004), http://www.omg.org/technology/documents/corbafacilities_spec_catalog.htm.
23. Object Management Group, Catalog of Corba Services specifications (2004), http://www.omg.org/technology/documents/corbaservices_spec_catalog.htm.
24. Object Management Group, Catalog of Corba/IIOP specifications (2004), http://www.omg.org/technology/documents/corba_spec_catalog.htm.
25. OSGi Alliance, OSGi Service Platform Release 3 (2003), http://www.osgi.org/.
26. Parlay group, Parlay X Web Services Specification 1.0 (2003), http://www.parlay.org/specs/index.asp.
27. Pokraev, S., Costa, P. D., Pereira Filho, J. G., Zuidweg, M., Koolwaaij, J. W., van Setten, M.: TI/RS/2003/137 Context-aware services: state-of-the-art (2003), https://doc.telin.nl/dscgi/ds.py/Get/File-27859/Context-aware_services-sota,_v3.0,_final.pdf.
28. The OWL Service Coalition, OWL-S: Semantic Markup for Web Services (2004), http://www.daml.org/services/owl-s/1.0/owl-s.html.
29. UDDI.org, UDDI specification version 3 (2004), http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3.
30. van Setten, M., Pokraev, S., Koolwaai, J., Context-Aware Recommendation in the Mobile Tourist Application Compass (2004), in Adaptive Hypermedia 2004, Eindhoven.
31. W3C, OWL (2004), http://www.w3c.org/2001/sw/WebOnt/.
32. W3C, Web service description language (2004), http://www.w3.org/TR/wsdl20/.