Personalizable Service Discovery in Pervasive Systems

Korbinian Frank
German Aerospace Center (DLR)
Institute of Communications and Navigation
Oberpfaffenhafen
82234 Weßling, Germany
Email: korbinian.frank@dlr.de

Vincenzo Suraci
Department of Computer and Systems Science
University of Rome ”La Sapienza”
Via Eudossiana, 18
00184 Rome, Italy
Email: vincenzo.suraci@dis.uniroma1.it

Jelena Mitic
Siemens AG
Corporate Technology
81730 Munich, Germany
Email: jelena.mitic@siemens.com

***Abstract*—Today, telecom providers are facing changing challenges. To stay ahead in the competition and provide market leading offerings, carriers need to enable a global ecosystem of third party independent application developers to deliver converged services. This is the aim of leveraging a open standards-based service delivery platform. To identify and to cope with those challenges is the main target of the EU funded project IST DAIDALOS II. And a central point to satisfy the changing user needs is the provision of a well working, user friendly and personalized service discovery. This paper describes our work in the project on a middleware in a framework for pervasive service usage. We have designed an architecture for it, that enables full transparency to the user, grants high compatibility and extendability by a modular and pluggable conception and allows for interoperability with most known service discovery protocols. Our Multi-Protocol Service Discovery and the Four Phases Service Filtering concept enabling personalization should allow for the best possible results in service discovery.**

I. Introduction

Nowadays computer networks offer innumerable possibilities in terms of accessible contents and services (e-commerce, telemedicine, web-TV) through the Internet. Often users are interested in locating a specific ”resource” that satisfies their requirements such as a file, a document, a multimedia clip, but also a printer, or a screen with dimensions greater than that one of a mobile phone. Also, network devices requires dynamic location of network services, such as content adaptation nodes to realize necessary codec conversion for setting up of a multimedia session. Finally, it is often requested that a wireless network can be established automatically and operate in an autonomous fashion, in other words, they should support advanced auto-configuration capabilities. As a consequence whenever services are to be located in a LAN or in the Internet, Service Discovery has to be performed. There is a general interest in the design of mechanisms for dynamic location of services and network resources in the research community in order to support computer network capabilities typical of a pervasive network environment. In other words, the fundamental problem is the investigation into service discovery architectures for pervasive computing systems. This problem can be further subdivided into two sub-problems:

(i) protocols or physical search mechanisms,
(ii) a service description architecture.

In order to enable automatic Service Discovery mechanisms, some protocols and software architecture have been proposed (SLP [1], JINI [2], UPnP [3], Salutation [4], UDDI [5], SDP [6], etc.). Compatibility, autonomy, and simplicity of service discovery are important requirements for applicability in pervasive systems. Description of a service with the maximum precision is of vital importance. Namely, if one is not able to discover exactly what is being looked for, the proposed, not perfectly fitting results are likely to be not used either. Therefore personalization has to come into play to assure individually matching search results. Another issue which is tightly coupled with service description concerns the vocabulary which is used for research, which should be as much standardized as possible, so that the domain of the research can be as large as possible. As a solution to this problem, an ontology based description model can be adopted, guaranteeing maximum flexibility in service description as well as a standardized vocabulary.

In this work we propose a general architecture which is independent from the particular Service Discovery Protocol, can perform Service Discovery both in automatic or manual mode, taking into account the context of a pervasive system in which the user is involved in as well as personal preferences of the future service user, and considering a semantic description of the available services.

In the remainder of this paper, we will put our work into context by discussing related work in the next section. In section 3, we will identify requirements and describe our Service Discovery architecture, before we describe more in detail the integration of Personalization in section 4. Afterward, in section 5, we will validate our framework against the described requirements and finally section 6 will give a conclusion.

II. Related Work

Protocols for service discovery choose between directory- and non-directory-based models. In the directory-based model a directory maintains service information and processes queries and announcements. Architectures can implement flat (like [7]) or hierarchical (e.g. [8] or [9]) structure. On the other side, when a query arrives in non-directory-based frameworks, every service processes it. If the service matches the query, it replies. When hearing a service announcement, a client can record service information for future use.

In the last years, more and more research work is spent on context-aware service discovery. As one of the first, a Jini ([2]) service query could specify a physical location as an attribute. Among others [10], [11], [12], [13], [14], [15], [16] and [17] are presenting further approaches, partly ontology based (e.g. [10]) and party based on key-value pairs as models of context information with or without relations between entities ([11], [13], [14], [16]). Integration or addition (quite no integration of [11], [13], [14]) of context data (mainly only static) to service offers is still handled very heterogeneous, which has consequences for the evaluation of the discovery process. Only [14] and [17] of the mentioned approaches take user preferences into account, while [10] allows to integrate user-defined attributes like ”nearby” into an service request. Advanced concepts for context-aware request modification come from the field of database systems [18], but aren't directly adaptable to service oriented architectures ([17]).

The Semantic Web approach argues that semantic annotation of resources and services is the key to support automatic discovery, interoperation, composition, etc. of Web services. This semantic annotation takes the form of an ontology specification, and an instance document conforming to the ontology describing the actual service. There are two major ontology based approaches to specifying composition, OWL-S [19] and WSMO [20]. Several European projects have been driving use of the WSMO framework. The DIP FP6-2004-507483 project [21] has been focusing on infrastructure necessary for the WSMO based semantic services. The INFRAWEBS FP6-2003-511723 project [22] has focused on the development of a software tool set for creating, maintaining and executing WSMO based Semantic Web Services.

These approaches cover some important aspects for service discovery, but omit or neglect other ones. Dynamics and personality of context usage is one, compatibility with other protocols and vocabularies another one that is not reflected satisfyingly in the known approaches. Also deployment issues still have to be taken into account as to minimize unnecessary computation on clients, servers and directories.

III. Service Discovery

The following paragraphs in section III will describe the general approach used for Service Discovery focusing in III-B our Multi-Protocol Service Discovery and different Service Filters we will deploy in section III-C after discussing the general design principles.

A. Design principles

Regarding the state of the art for service discovery in existing systems compared with our goals for real pervasive computing, we identified the following overall targets for the DAIDALOS Service Discovery Middleware:

1) Transparency:
   The main purpose of DAIDALOS II architecture is to make the service discovery process transparent to the user and to the rest of the software architecture.
2) Compatibility:
   At the same time, the discovery process should be backwards compatible with the plethora of standardized discovery protocols.
3) Interoperability:
   Thus it is necessary to make all these protocols capable of inter-operating in order to provide unified service discovery functionality with a unique interface.
4) Extendability:
   Just as our middleware has to provide the means for backward compatibility, it must allow for new, future developments to be integrated. Therefore the architecture must be modular, extensible and – if possible – hot-pluggable.
5) Personalization:
   Finally, the service discovery should return only promising service candidates, i.e. such service offers that are likely to be used by the individual triggering the discovery. Therefore semantic information of the service offers have to be taken into account and matched against the user's needs brought into the query either manually or automatically.

Following these considerations, we designed a service framework that should be able to cope with all the challenges. The generic discovery approach addresses these needs with the architecture shown in figure 1. It is sufficiently general to allow each existing discovery protocol to be correctly used. Whenever the *Generic Discovery Module (GDM)* is used for some service request, the incoming query string is prepared and enriched by the *Query Preprocessor (QP)*. Then it is properly translated to the available service discovery protocols, thanks to the presence of a *Pluggable Discovery Module (PDM)* in charge of mapping the generic discovery requests into the specific protocol requests and executed in the native

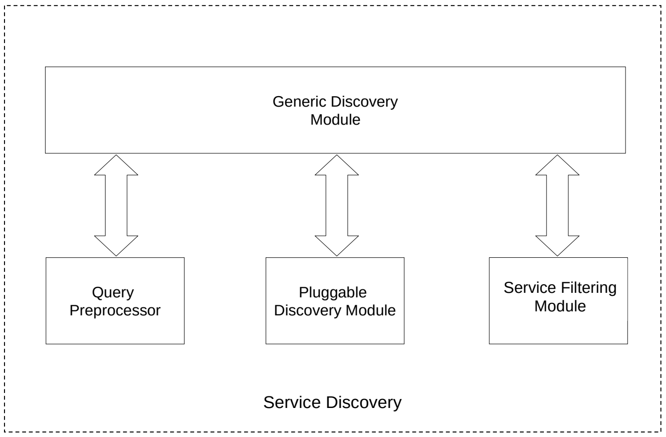


Fig. 1. Service Discovery Reference Architecture ([23])

discovery protocol. After this step a series of optional filters in the *Service Filtering Module (SFM)* are enhancing the results before they can be returned to the requester.

B. Multi-Protocol Service Discovery

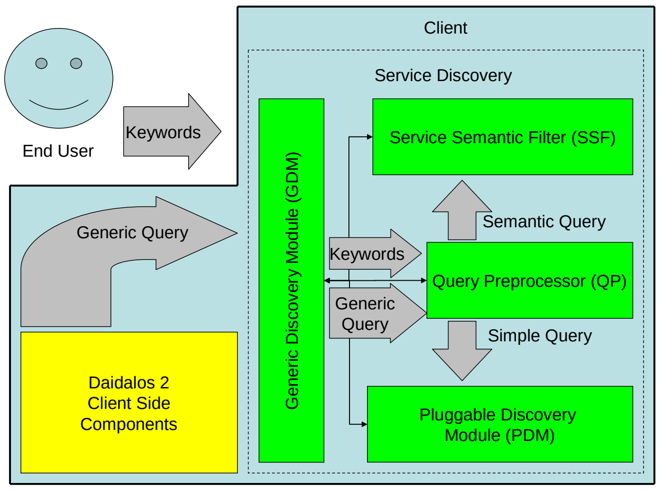
To enable the discovery of services offered in different *service discovery protocols (SDP)*, we have to provide a *Generic Discovery Query Language (GDQL)* that abstracts from the concrete syntax of a single protocol, but is still translatable into the concrete syntax. In the following paragraphs we will detail on how this target is realized from an implementation point of view, revealing parts of the Java implementation.
The GDQL developed for DAIDALOS is composed by two types of discovery query languages:

1) *Common Discovery Query Language (CDQL)* – this is a SQL-like and common to the different low-level service discovery protocols (like SLP, UPnP, etc.), providing basic discovery/registration functionalities. This language is used by the PDMs in order to create a protocol specific query that will be used to perform a basic service filtering. The common discovery query language is used to create `Simple Queries`.
2) *Semantic Discovery Query Language (SDQL)* – that is able to handle ontologies and to filter services in a semantic manner. It is also in charge of taking into account potential user requirements on the context of the services and the surrounding environment. The semantic discovery query language is used to create `Semantic Queries`. In DAIDALOS 2 the *RDF Data Query Language (RDQL)* will be used as SDQL.

In figure 2 you can see the process how this query languages will be used. Originated by a DAIDALOS system component in a Generic Query or by an end user giving keywords the query is forwarded to the Query Preprocessor. The QP is providing the `ICreateQuery` interface which the GDM uses to obtain a `Generic Query`. A Generic Query is a pair $<$ `Simple Query`, `Semantic Query` $>$ where:

- `Simple Query` – it is expressed in a CDQL and based on simple filtering mechanisms (e.g. the type of a service or a list of attribute-values pairs).
- `Semantic Query` – it is expressed in a SDQL (like RDQL for instance) and aware of semantic based filtering mechanisms.

The `Simple Query` is translated into a discovery protocol by the chosen PDM and the resulting list of services is filtered by the SFM based on the Semantic Query.
The CDQL has been designed within DAIDALOS II to be at the same time powerful enough to perform basic discovery/registration mechanisms and to deal with the different protocol specific characteristics. Due to these requirements, the CDQL syntax is very similar to SQL's one. To better understand what CDQL looks like, we will propose an example in the following. An end user is looking for ink-jet printers that support CMYK or RGB, print at least 10 ppm, cost less than 1SS/page, have a maximum of 10 documents in its queue, are



Client
Service Discovery
End User
Keywords
Generic Query
Daidalos 2 Client Side Components
Generic Discovery Module (GDM)
Service Semantic Filter (SSF)
Semantic Query
Keywords
Query Preprocessor (QP)
Generic Query
Simple Query
Pluggable Discovery Module (PDM)

Fig. 2. Discovery Query Language diagram ([23])

less than 50 meters away (from the user starting the discovery procedure) and inside his building.
The corresponding CDQL query would be:

```
SELECT *
FROM   local
WHERE  ServiceType = printer
       AND ColourDepth = four_colour_process
       AND (ColourCombination = CMYK
              OR ColourCombination = RGB)
       AND PrintingSpeed >= 10
       AND PrintingCost <= 1
       AND Distance <= 50
USING  UPnP OR SLP
```

C. Four Phases Service Filtering

In this section, we present four basic filters that should be mandatory to be used in ubiquitous service environments. A service filter thereby is a module that applies criteria to service advertisements and discards those which do not meet the criteria. As such, Service Filtering is an important part for a large scaled system enabling pervasive services. It reduces the number of service candidates because of its destructive character. If no filtering is applied to the list of discovered services in a "pervasive world" with thousands of available commercial and private services, they would all have to be processed in subsequent stages like e.g. Service Ranking.

Input to this filtering process is a completed query which is the output of the Query Preprocessor that will be described more in detail in section IV-A.

1) Service Type Filtering: The Service Type Filtering is directly connected with the concept of service discovery and in our scenario often already performed by the pluggable Multi-Protocol Service Discovery. If the low level service discovery protocols (like SLP or UPnP) are understanding the concept of `ServiceType`, then this task is expressible already in a CDQL. But in case low level protocols don't contain service type information or too few details/variations (like Bluetooth profiles for instance), a special filter is needed to extract this information from a semantic service description. This is a basic filtering process in which just a small set of information is necessary in order to identify a preliminary set of available services that will be further filtered in the next phases.

2) Semantic Filtering: The Semantic Filtering (also Ontology-based Filtering) deals not only with the protocol given service description, but with the semantically enhanced service descriptions based on OWL-S. Like that, many more attributes of a service are known and can be subject to filtering constraints as already described in [24]:

- User-dependent constraints:
  For instance lists of keywords provided by the user for the query are representing filtering criteria.
- Service-dependent constraints:
  Filtering criteria may be imposed by the services (or more precisely: their descriptions) as well as by the users. Services can have limitations, e.g. with respect to resources or compositions with other services. Particularly well known are (service-dependent) criteria concerning the user's location (also known as the service's scope).

3) Contextual Filtering: As can be seen in the state of the art in section II, the current service discovery architectures are usually based on static information and are not aware of the surrounding environment and the user context. In our view on the other hand, it is not acceptable that a future service management architecture ignores context information which would make the system aware of the world's dynamics.

In order to introduce the concept of a context aware service discovery, we will shortly present our concepts of context requirements and context information (for a more detailed description see [25]). There are three main actors that play an active role in a context aware service discovery, namely end user, service and environment. Table I will show their specific view on context.

Without entering in detail, we assume that both Context Information and Context Requirements will be described using ontologies and all context information issues will be managed by a Context Manager (for more details see [26]) that stores context information produced by the context sources, delivers context information to external applications that need it, provides filtering policies to select the context information to obtain and offers a Publisher-Subscriber mechanism to register for specific context information. The overview of the overall context management architecture is depicted in Fig. 3 below.

In the following we will have a deeper look into the concrete interactions between the entities involved in a general service discovery process in case of service registration and discovery:

In order to have a context aware service registration, whenever a *Service Agent (SA)* wants to advertise a service, it registries the service storing into the *Directory Agent (DA)* the context information about the service requirements on the end user, the service requirements on the environment and a pointer to the service context (maintained by the context manager) – all the information that can be seen in table I. Then a *User Agent (UA)* searches for a service and sends with its query the context information about user requirements on service, environment and also a pointer to the user context (maintained by the context manager). If finally the DA receives a service query from the UA, it checks if the registered service profile matches the service query, if the service and the environmental context match the user requirements, as well as if the user and the environmental context match the service requirements. So, a service is filtered if either one ore more of the context requirements or the query are not satisfied.

4) Privacy-based Filtering: In DAIDALOS II each legal person has one or more VIDs (see again [27]). Thus, a service provider also has a set of VIDs and whenever he advertises a service in a specific network, that service is associated to a specific VID. Network operators, but also worried parents for instance, might not want to make all services discoverable. In

TABLE I
CONTEXT FROM DIFFERENT POINTS OF VIEW

	End User	Service	Environment
Context Information examples	terminal battery charge, user position, terminal screen size, etc.	service location, service operational status, etc.	Whatever surrounds the end user and the service.
Context Source examples	portable GPS systems, etc.	location, etc.	environmental sensors, network measures, etc.
Context requirements	require specific Context Information from either services or the environment. Examples could be the distance from the service, network availability, etc.	needs regarding Context Information from the end user or the environment. For instance this might be a certain screen size, network QoS or environmental conditions	

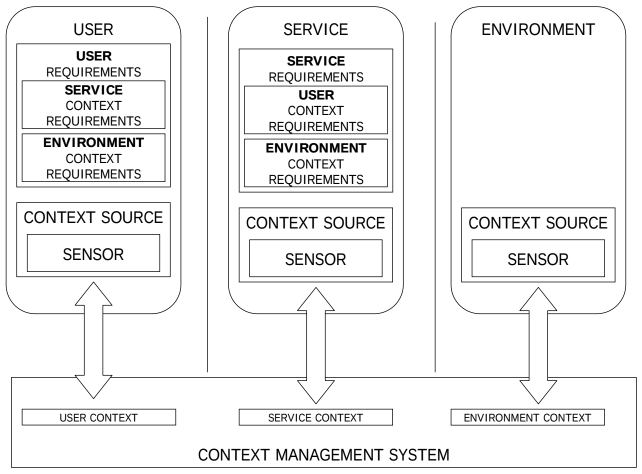


Fig. 3. Overview of the Context Management Architecture ([23])

order to avoid that any registered service could be discovered by any person, the VID of the Service Provider can be associated to a certain number of groups (or communities). Like that, our framework could guarantee that only members of those specified communities will be able to discover the service, because the requester's group memberships can be identified by checking his VID.

In order to allow such a mechanism to work properly, a specific privacy filtering phase is required. Whenever an end user performs a service discovery, the list of discovered services (i.e. matching the user query, the user preferences on service discovery and the user context) will be finally filtered taking into account also the privacy issues as mentioned above.

It is clear that a Service Provider could have more than one VID, so he can differentiate the availability of its own services to different kinds of user communities. Moreover each single VID belonging to a service provider or an end user could be associated to none, one or more communities. Instead a registered service belongs only to one VID, the one used by the service provider to register the service.

IV. Personalization of discovery results

As discussed earlier on, pervasive and in particular context aware computing aim to ease things for users. Therefore services should be selected following the user's preferences. As we don't assume that each user always wants to enter (or even is aware of) all his preferences and constraints, in DAIDALOS we have built a Personalization subsystem, both storing and learning user preferences for each subscriber of the system (see [28]). So either the user enters his preferences manually or they are integrated automatically by the system using an interface of the Personalization subsystem.
We distinguish between two types of preferences:

- *Service Discovery Preferences* which are *hard* criteria (”Don't”, ”Never”, ”Always”, ”Only”,...) representing the user's constraints like for a Restaurant-Finder service:
  `Don't show Chinese Food.`
- *Service Selection Preferences* that are just giving the user's bias (*soft* criteria) and are used in a later Service Ranking process, see the document [23] for more information on that.

Both types of preferences should be handled in different stages of the service discovery and selection process. Though all user preferences could be treated *after* a first basic discovery of all available services, it does make sense to reduce the number of service candidates as early as possible as already discussed in III-C. Therefore we integrate Service Discovery Preferences by the *Query Preprocessor* into the query that is already executed by the various filters, before we apply the Service Selection Preferences to the resulting list in a component called *Service Ranker*. The tasks of these two components will be presented in the following subsections

A. Query Preprocessing

Before the actual Service Discovery is executed in the filters, the DAIDALOS middleware must make sure that a well-formed, valid and complete query statement is available. This is the task of the Query Preprocessor which first personalizes the query and then translates all manually given or system-stored constraints into one query-statement.

a) Completeness of the information included into the query: The completeness of the information relates mainly to adding further user preferences to the key words given by the user to start the query. Those user preferences are retrieved from the DAIDALOS Personalization subsystem (see [29]), because it is sensible to store them permanently in his user profile, whenever a user tends to reuse his constraints on service queries. The information available in a user profile can be applied every time the user queries the system.
Retrieved information have then to be compared with the received keywords to avoid contradictions that would result in empty result sets for the query. Manually entered constraints have to get priority in the case of conflicts.

b) Formulation of the query in a common discovery query language: After this completion stage, the QP has a second important task: a well-formed and valid query has to be built, that is able to interact with all the different service discovery protocols used in the pluggable Service Discovery architecture of DAIDALOS.
This complete and ”translated” query is then fed back to the GDM and used further in SFM and PDM.

B. Service Ranking

The Service Ranker is the next component in the DAIDALOS architecture after the basic Service Discovery depicted in Fig. 1. It is receiving from the GDM an *unsorted* list of service candidates. Unlike the Query Preprocessor, the Service Ranker is not only retrieving the data from the Personalization subsystem, but also applying them. For retrieval it can use the same interface only specifying a different type of preferences.
Typical examples for Service Selection Preferences would be ”`Cheaper restaurants preferred`” or ”`I prefer to sit outside`”. Their application to the list of service candidates depends on their system internal representation. A simplistic approach would be hierarchical $<$`ServiceType`, `ServiceAttribute`, `value`$>$ (e.g. $<$`Restaurant`, `Cost`, `minimum`$>$) combined with priorities in order to overcome conflicts of different preferences. More sophisticated ranking could be based on evaluation of objective functions, e.g. in Bayesian Decision Networks that are already partially applied in the DAIDALOS Context and Personalization subsystems.

V. Validation

Our middleware is based on a Plug&Play approach: the Pluggable Discovery Modules can host different service discovery protocols, making it easy to expand the users' possibility to discover services belonging to different technologies and to be compatible with many different service protocols. The use of ontologies to describe the services in a semantic way and the use of standard languages helps the architecture

to inter-operate with whatever type of service, but also to be easily merged with existing service management solutions. With well defined interfaces (QP and Service Ranker) it is allowing for personalization of each query, while the user has only a very simple interface for keywords and the whole process stays completely transparent to him.
These considerations imply that the requirements identified in section III-A have been successfully matched by the proposed architecture.

Within DAIDALOS, the architecture is implemented as a prototype and used in three scenarios. The Java implementation is running in an OSGi environment on a IPv6 network. Service Descriptions are based on an extended OWL-S format, which we manipulate with the Jena2 toolkit. We realized the components in a thin-client approach. Only basic functionality is running on the mobile device preserving the performance of the discovery process, while the computationally hardest part (filtering and ontology reasoning) is shifted to the back-end.

VI. Conclusion

In this work we presented a general and innovative architecture for service discovery in Pervasive Systems. We particularly discussed the main design principles and presented the overall structure of the architecture.
The architecture presented here provides a general and innovative way to perform personalized service discovery based on a four phase filtering process, where service-type, semantic, context-aware and privacy filters are applied together with user specific preference information to find the best services matching the search criteria.
This middleware is applicable to many different scenarios in which the service discovery process is necessary and needs to be easy to install and easy to use. Thanks to its capabilities this platform could transform a simple set of services in a real pervasive environment in which the end user feels to be involved in.

Acknowledgments

The work described in this paper is based on results of IST FP6 Integrated Project DAIDALOS. DAIDALOS 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. The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.
The authors would like to thank members of DAIDALOS team who partially supported this work by participating to service discovery issues in a context of pervasive computing environment.

References

[1] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," United States, 1999.
[2] "Jini Architectural Overview. White Paper," Sun Microsystems, 1999. [Online]. Available: http://www.sun.com/jini/whitepapers/architecture.pdf
[3] UPnP Forum, "UPnP device architecture 1.0," July 2006.
[4] B. A. Miller and R. A. Pascoe, "Salutation service discovery in pervasive computing environments," IBM White Paper, Tech. Rep., Feb. 2000. [Online]. Available: http://www-3.ibm.com/pvc/tech/salutation.shtml
[5] T. Bellwood, L. Clement, and C. von Riegen, "UDDI specification version 3.0.1." UDDI Spec Technical Committee, Tech. Rep., October 2003.
[6] "Specification of the Bluetooth System v1.1," 2001. [Online]. Available: http://www.bluetooth.com/dev/specifications.asp
[7] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *SOSP*, 1999, pp. 186–201.
[8] Apple Computer, "Bonjour release: 107.6 (formerly "rendezvous")," 2006.
[9] J. R. von Behren, E. A. Brewer, N. Borisov, M. Chen, M. Welsh, J. MacDonald, J. Lau, and D. E. Culler, "Ninja: A framework for network services," in *USENIX Annual Technical Conference, General Track*, 2002, pp. 87–102.
[10] T. Broens, S. Pokraev, M. van Sinderen, J. Koolwaaij, and P. Dockhorn Costa, "Context-aware, ontology-based service discovery," in *EUSAI*, 2004, pp. 72–83.
[11] C. Doulkeridis, N. Loutas, and M. Vazirgiannis, "A system architecture for context-aware service discovery." [Online]. Available: citeseer.ist.psu.edu/doulkeridis05system.html
[12] S. Penz, "SLP-based service management for dynamic ad-hoc networks," in *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*. New York, NY, USA: ACM Press, 2005, pp. 1–8.
[13] P.-G. Raverdy, O. Riva, A. de La Chapelle, R. Chibout, and V. Issarny, "Efficient context-aware service discovery in multi-protocol pervasive environments," in *MDM '06: Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)*. Washington, DC, USA: IEEE Computer Society, 2006, p. 3.
[14] S. Cuddy, M. Katchabaw, and H. Lutfiyya, "Context-aware service selection based on dynamic and static service attributes," in *Wireless And Mobile Computing, Networking And Communications, 2005.*, 2005, p. 4.
[15] S. Marti and V. Krishnan, "Carmen: A dynamic service discovery architecture." [Online]. Available: citeseer.ist.psu.edu/marti02carmen.html
[16] M. Samulowitz, F. Michahelles, and C. Linnhoff-Popien, "CAPEUS: An architecture for context-aware selection and execution of services," in *DAIS*, 2001, pp. 23–40.
[17] F. Klan, "Context-aware service discovery, selection and usage," in *Grundlagen von Datenbanken*, 2006, pp. 95–99.
[18] A. Lubinski, *Mobile Datenbanken und Informationssysteme - Konzepte und Techniken*. dpunkt.verlag, 2005, ch. Anfragen mobiler Benutzer, pp. 93–98.
[19] "OWL-S," http://www.daml.org/services/owl-s.
[20] "WSMO," http://wsmo.org.
[21] "IST project DIP," http://dip.semanticweb.org, 2004.
[22] "IST project INFRAWEBS," http://www.infrawebs.eu, 2003.
[23] R. Mullins, M. Crotty, J. Mitic, K. Frank, P. Robertson, R. M. Svedsen, B. Farshchian, and V. Suraci, "Pervasive service manager," IST Daidalos II, Tech. Rep., December 2006.
[24] S. Mignanti, V. Suraci, and C. Di Menna, "An ontology-based multi-protocol service discovery framework," in *Mobile and Wireless Communications Summit, 2007. 16th IST*, 2007, pp. 1–5.
[25] V. Suraci, S. Mignanti, and A. Aiuto, "Context-aware semantic service discovery," in *Mobile and Wireless Communications Summit, 2007. 16th IST*, 2007, pp. 1–5.
[26] M. Strimpakou, I. Roussaki, C. Pils, M. Angermann, P. Robertson, and M. E. Anagnostou, "Context modelling and management in ambient-aware pervasive environments." in *LoCA*, 2005, pp. 2–15.
[27] W. Fitzgerald, K. Doolin, F. Mahon, C. Hauser, A. F. Gomez-Skarmeta, S. Butler, P. Schlosser, and B. Weyl, "Daidalos security framework for mobile services," in *Proceedings of eChallenges 2005*, June 2005.
[28] S. McBurney, E. Papadopoulou, N. Taylor, H. Williams, K. Frank, P. Robertson, G. L. Bello, and M. L. Demarie, "Architecture and design: Personalisation and learning management," IST Daidalos II, Tech. Rep., December 2006.
[29] Y. Yang, M. H. Williams, R. Pooley, and R. Dewar, "Context-aware personalization in pervasive communications." in *ICEBE*, 2006, pp. 663–669.