

# Efficient Context-aware Service Discovery in Multi-Protocol Pervasive Environments

Pierre-Guillaume Raverdy\*, Oriana Riva<sup>1\*\*</sup>, Agnès de La Chapelle\*, Rafik Chibout\*, Valérie Issarny\*

\*INRIA-Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay, France,  
firstname.lastname@inria.fr

\*\*Helsinki Institute for Information Technology, P.O. Box 9800, FIN-02015 HUT, Finland  
oriana.riva@hiit.fi

## Abstract

*Service discovery is a critical functionality of emerging pervasive computing environments. In such environments, service discovery mechanisms need to (i) overcome the heterogeneity of hardware devices, software platforms, and networking infrastructures; and (ii) provide users with an accurate selection of services that meet their current requirements. To address these issues, we have developed the Multi-Protocol Service Discovery and Access (MSDA) middleware platform<sup>2</sup>, which provides context-aware service discovery and access in pervasive environments. This paper primarily focuses on the design and implementation of the context-awareness support of MSDA. Context-awareness not only provides a more accurate service selection, but also enables a more efficient dissemination of service requests across heterogeneous pervasive environments. We present the design and prototype implementation of MSDA, along with experimental results that demonstrate the advantages derived by introducing context awareness.*

## 1. Introduction

In pervasive environments, people are surrounded by a variety of computing devices offering services of different types. As users need to discover and interoperate with such services, service discovery represents a crucial functionality. In order to be effective, service discovery mechanisms need to cope

with the heterogeneity inherent to pervasive environments, and be accurate enough to provide users with a selection of relevant services.

Pervasive environments are heterogeneous both in terms of networking infrastructures and interaction protocols. Network heterogeneity arises from the use of various wireless technologies (e.g., cellular networks, WiFi, or Bluetooth), from the adoption of different management models (e.g., infrastructure- or ad hoc-based), and from the variable support for IP-level communication and configuration functionalities (e.g., IP multicast, DHCP). This heterogeneity leads to many independent networks being available to users at a location. As users can only be connected to a limited number of networks at the same time (often a single one), many services are often not accessible (i.e., not IP reachable). The heterogeneity of interaction protocols arises from the concurrent use of different middleware platforms (e.g., Jini, UPnP, Web Services). Due to incompatible data representation and communication formats, these protocols do not interoperate with each other. Hence, users are able to discover only services that are advertised with the protocol(s) they support.

Moreover, traditional service discovery protocols focus on capturing in their service advertisements and queries only the static characterization of services and clients. They do not take into account the user's current needs and situation, such as user's location, type of terminal in use, or control policies. In service-rich environments, such an approach performs highly inefficiently by providing users with long lists of services where only few actually meet the current requirements in terms of service quality, performance, cost, accessibility, and so forth. Guaranteeing more accuracy in the selection of available services is especially beneficial to mobile users, who are often equipped with resource-constrained devices.

<sup>1</sup> While visiting the ARLES project at INRIA.

<sup>2</sup> This work is part of the IST UBISEC project and has been funded by the European Commission, FP6 contract number 506926, <http://www.ubisec.org/>.

To address the above issues, we have been developing the *Multi-Protocol Service Discovery and Access* (MSDA) middleware platform that integrates existing Service Discovery (SD) and Service Access (SA) protocols. MSDA enables clients to discover and access services available in the entire pervasive environment, while still using their preferred service discovery and access protocols. MSDA exploits context information to provide both intelligent disseminations of discovery requests over heterogeneous networks (e.g., propagation to less congested networks) and context-aware service selection (e.g., select nearby services). In this paper, we focus on describing the context-awareness support that MSDA offers, while a detailed description of how MSDA addresses heterogeneity and interoperability issues can be found in [11].

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 briefly describes the architecture of MSDA. Section 4 focuses on the context-based features of MSDA and Section 5 evaluates the MSDA prototype implementation in terms of advantages/overhead introduced by supporting context-awareness. The paper concludes in Section 6.

## 2. Related Work

Over the years, many academic and industry-supported Service Discovery Protocols (SDPs) have been proposed (e.g., UPnP [13], or SLP [7]). One common limitation of these SDPs is that they were initially designed based on specific assumptions about the underlying network (e.g., Internet, home networks), the users' behavior, or the applications' needs. While efficient for the targeted environment, existing SDPs prove to be inefficient or not applicable in different settings. A further limitation of existing SDPs is that they support only primitive service matching based on exact matching of static and predefined attributes, without taking into account the user's context information [4]. Relevant contextual information includes, for example, the situation of the user (e.g., location, activity, interests), the capabilities and status of the employed terminal (e.g., CPU status, screen resolution), or the computing environment (e.g., network connectivity, available bandwidth). However, the diverse environment constraints and the de-facto standard status of some of the existing SDPs make it unlikely for a unique SDP to emerge.

Very few projects have investigated how to practically integrate context-aware service selection support in existing service discovery protocols, such as Jini, Salutation, or UPnP. Lee and Helal [9] extended Jini to support context-awareness by means of dynamic

context attributes in the service description (i.e., during service registration). Clients generate discovery requests in the original Jini format using only static attributes. Relevant service descriptions are selected based on the static attributes, and on the context attributes that are dynamically calculated by the lookup service. A key assumption behind this approach is that the context of the registry used by the client approximates the context of the client. In our approach, we do not integrate context directly into one (or more) specific SDP, but we transcode information provided by any active SDP into a common format, which is then extended with various types of context information. We integrate context information characterizing not only services, but also users and networking environment, thus providing a more comprehensive usage of context information.

Many projects have also proposed new context-aware service discovery protocols for infrastructure-based and ad hoc networks, and in particular location-aware SDP such as Splendor [14]. Lee [10] proposes an agent-based approach that aims to transparently and constantly provide users with the best networked service based on their preferences and needs. Solar [2] is a middleware platform that enables context-aware applications to locate environmental services based on the current context. Q-CAD [1] is a QoS and context-aware resource discovery framework for pervasive environments. Each application dynamically specifies in its profile context parameters to be taken into account when discovering resources, and also specifies a utility function to be maximized when binding the most suitable resources. In addition to supporting existing SDPs, our approach differs from these projects as we use context information not only for selecting services of interest, but also for controlling the propagation of service requests across multiple heterogeneous and often resource-constrained networks. This permits us to better preserve available network resources, which is essential in pervasive and mobile environments.

## 3. Overview of MSDA

MSDA is a lightweight middleware platform that facilitates service discovery and access in pervasive environments. We model pervasive environments as a dynamic composition of heterogeneous IP networks belonging to different administrative domains. In general, we assume that global IP routing is not supported. Moreover, the Internet and cellular networks are considered as conduits enabling the interconnection of the various networks. Figure 1 illustrates such a composition, with a user in a music

store able to access an ad hoc network and two hotspots. Some devices in the ad hoc and hotspot networks are connected through cellular networks and the Internet to their intranets and home networks.

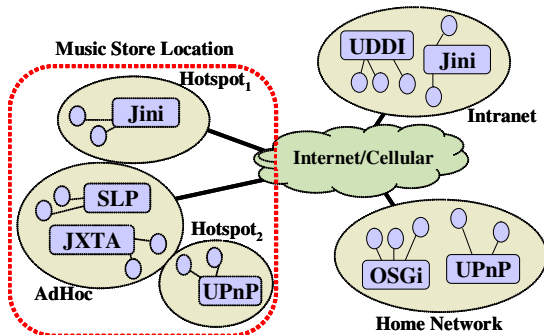


Figure 1: MSDA Pervasive Environment

### 3.1. Design Principles

To support service discovery and access in pervasive environments, MSDA needs to address the following two issues. First, clients and services in different networks cannot interact with each other due to network limitations (e.g., no global IP routing, firewalls, private IP addresses). Second, clients and services using different service discovery platforms cannot interoperate.

To address the above issues, we designed MSDA based on the following architectural principles:

**Middleware integration:** MSDA is an additional layer on top of existing middleware platforms that enables clients to interact with services in different discovery domains (i.e., advertised by different SD protocols).

**Dynamic configuration:** MSDA is instantiated independently in each network. MSDAs in nearby networks communicate with each other to disseminate service information and provide remote service access. Each MSDA instance independently selects to which neighboring networks to connect to.

**MSDA as a service:** Each instance of MSDA registers as a service, namely the *MSDA Service*, in all active discovery domains in its network. The MSDA Service provides interfaces for service discovery and service access. Applications discover and interact with the MSDA Service using their preferred discovery and access protocols.

### 3.2. MSDA Components

As Figure 2 shows, MSDA includes the following components: (i) the *MSDA Manager* that provides service discovery and access to clients within the

network; (ii) *Service Discovery and Access (SDA) Plugins* and *Transformers* that interact with specific discovery domains to collect service information and perform service access (e.g., UPnP or SLP Plugins); and (iii) *MSDA Bridges* that assist MSDA Managers in extending the service discovery and access to services present in other networks of the whole pervasive environment. Specifically, service access is achieved through application-level routing provided by MSDA Bridges, and through transcoding of access messages. In this paper, we focus on the context-aware service discovery support provided by MSDA.

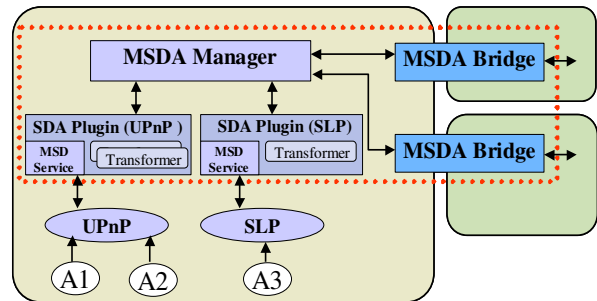


Figure 2: MSDA Platform Overview

In each network, clients on any device can interact with the local MSDA service. In addition, device owners can configure their devices to host MSDA (i.e., register as a potential MSDA Manager and/or MSDA Bridge). One MSDA Manager is dynamically elected from the potential devices, and the elected MSDA Manager dynamically activates the potential MSDA Bridges within its network.

### 3.3. MSDA Formats

Services in MSDA are described using the *MSDA Description* format. MSDA Descriptions are created by MSDA components (i.e., MSDA Managers and SDA plugins) based on the initial SD-specific service description. MSDA Descriptions consist of: (i) *creation information* that details the components that generated the description and where it was generated (i.e., IDs of MSDA Manager, SDA plugin and Transformers); (ii) *service information* that specifies the service properties as provided by the original SD protocol (e.g., service name, methods names and input/output parameters if available); (iii) *service context information* that specifies context properties and admission control policies associated to the service instance; and (iv) *propagation information* that describes the path (in terms of networks and MSDA Bridges) between the client and the service.

*MSDA Requests* carry the user's service discovery request. They include: (i) *service information* in which the client specifies the functional properties of the service to be discovered (e.g., service name or interface); (ii) *user context information* in which the client specifies the characteristics/status of the device and its owner, along with filters/preferences regarding how to carry out the service discovery; (iii) *processing information*, which defines how MSDA Managers (both local and remote) should process the request and return the results (e.g., results returned all at once); and (iv) *propagation information* that records the networks so far traversed by the request.

### 3.4. Service Discovery in MSDA

An MSDA-aware client looking for services in the pervasive environment first discovers the MSDA Service using its preferred SDP. The MSDA Service offers a pull-based interface for service discovery (i.e., the client sends a discovery request to the MSDA Service and waits for results). The MSDA Service passes the discovery request to the local MSDA Manager. In turn, the MSDA Manager forwards it to: (i) the local SDA Plugins for discovering suitable services in the local network, and (ii) the MSDA Bridges for propagating the request to MSDA Managers in nearby networks. The MSDA Managers receiving a remote discovery request process it as a local discovery request (i.e., forward it to local SDA Plugins and to MSDA Bridges), and return the results to the originating MSDA Bridge. Finally, the original MSDA Manager of the client collects local and remote results, and returns them to the client.

Each SDA Plugin collects service information in a specific discovery domain (e.g., SLP, UPnP) on behalf of the MSDA Manager. Depending on the SDP, the SDA Plugin either registers for service advertisements (i.e., push-based protocol), or directly performs service discovery (i.e., pull-based protocol). The SDA Plugin generates an initial MSDA Description based on the received SD-specific service information (service advertisement or result of a discovery request). This initial MSDA Description is then processed by the MSDA Transformers associated with the SDA plugin. Typically, MSDA Transformers are external components provided by companies, consortiums, or interest groups that can extend the initial MSDA Description with additional service information.

It should be noted that in the case of push-based SD protocols, the MSDA Descriptions provided by SDA Plugins are stored/cached in a local repository by the MSDA Manager. Therefore, for these push-based protocols, the MSDA Manager directly evaluates

incoming MSDA Requests against this repository (i.e., without forwarding it to the SDA plugins), and returns the matching service descriptions. It should also be noted that the propagation of the MSDA Request is done in parallel with its processing within each network (i.e., the maximum discovery time in the global network is equal to the maximum discovery time in a local network plus the maximum time for forwarding the request and returning the response).

## 4. Context-awareness in MSDA

Context support in MSDA aims at preserving available (and often limited) resources in the network and on (mobile) devices by avoiding unnecessary dissemination of service requests, and by providing users with services that best match their requirements.

### 4.1. Context Model

In MSDA, we consider *context information* of the networking environment, the interacting users, and the service instances. We model context information for these entities as the combination of *context parameters* and *context rules*. Context parameters correspond to static and dynamic attributes characterizing the entity. Context rules correspond to control policies expressing preferences, choices, and filters for the control of the discovery process in terms of resources to be used, level of security to be applied, type of services to be selected, and so forth.

The context of the networking environment aggregates information related to the specific network. It includes static context parameters (type of network, supported protocols, security level, etc.), dynamic context parameters (number of active users, number of available services, load, etc.), and admission control policies for incoming and outgoing messages. The context of the user includes information about the interacting user (i.e., person and device), and her control policies. Control policies express requirements on how the service instance should be selected and on how the discovery request should be propagated. At the current stage of our project, we have mostly focused in representing the capabilities and current state of the device in use (e.g., location, time, storage space, etc.). Finally, the service context aggregates static and dynamic information about a service instance (cost, supported image format, waiting queue length, etc.) along with admission/security policies for regulating service access (e.g., reject uncertified requests).

Context parameters are stored in an open table of tuples (i.e., name and value pairs). To model context parameters, part of the vocabulary borrows terms from

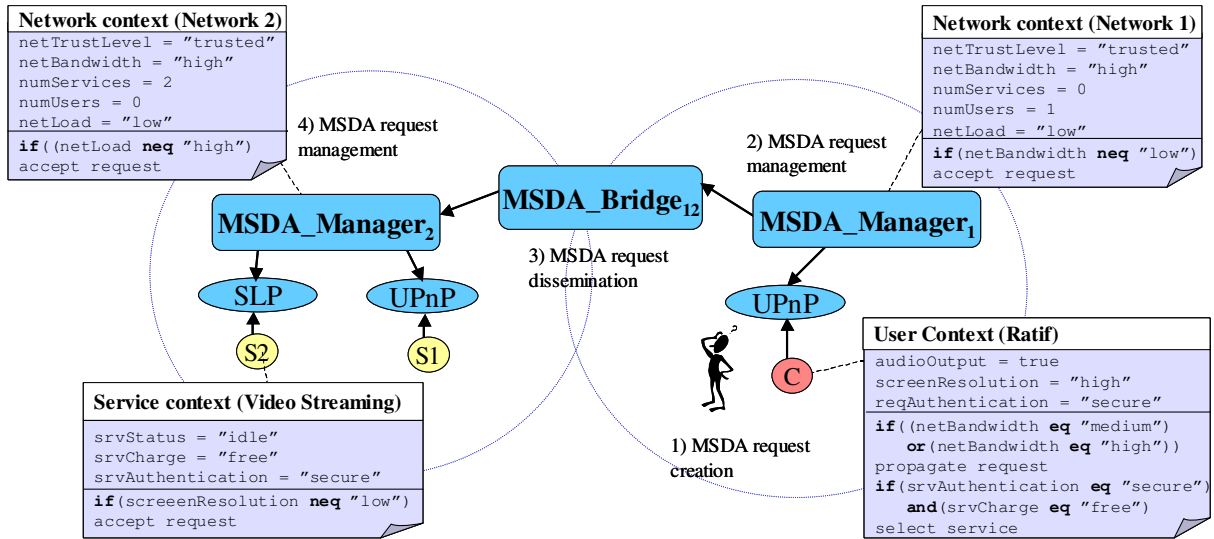


Figure 3: Example of context utilization in MSDA

consensus ontologies for pervasive environments such as SOUPA [3] or FIPA Device ontology [5].

Context rules are expressed in the form of condition/action statements. Conditions are articulated as Boolean expressions, and the operators currently supported are *and*, *or*, *equal*, *not-equal*, *more than*, and *less than*. Actions currently supported are *propagate request*, *accept request*, and *select service*. Each rule is set as optional or mandatory, as the context parameter required to evaluate the rule may not be available. For example, a network may not be able to monitor the number of active users. Moreover, *and* and *or* operators permit to flexibly combine elementary rules to build complex ones.

In order to illustrate our context model, examples of user, network and service contexts are depicted in Figure 3. The user context information associated with the discovery request sent by the user *Ratif* (Step 1) specifies that the user's device has an audio output and a high-resolution screen, and that the user is looking for free of charge and secure services in medium/high bandwidth networks. In *Network2*, the service context associated with the video streaming service specifies that this service is free of charge, secure, currently idle, and requires devices with medium/high-resolution screen.

## 4.2. Context Management

Various components participate to the provisioning of context information used in MSDA: MSDA Managers, users, MSDA Plugins, and Transformers.

MSDA Managers initially gather static context information about the networking environment, and subsequently monitor the network activity to

dynamically update context parameters such as number of active users, number of available services, current network load, and so forth. Moreover, the MSDA Manager uses special control messages, namely *context messages*, to periodically notify MSDA Bridges within the domain about network context information. Nearby network domains can further exchange their network context information to gain knowledge about larger networking environment.

Users specify their context parameters and rules in the MSDA Requests sent to the MSDA Service. We therefore assume that context monitors and device controllers are available on the user's terminal, and that the user is able to express her control policies in the form of semantically correct context rules.

Finally, SDA plugins and the associated MSDA Transformers provide the service context information, which is carried by the MSDA Description. When context information is already available within the initial service description, the SDA Plugin automatically converts it, and adds it to the MSDA Description. Additionally, MSDA Transformers collect service context information to extend the MSDA Description. An MSDA Transformer may have *a priori* knowledge of certain services or may dynamically query specific context APIs of the service itself (or any third-party service) in order to update service context information (e.g., availability status of the service). Typically, MSDA Transformers can be specified for interacting with a certain category of services (e.g., supplied by the same manufacturer) or for supporting a certain context interface (e.g., using a specific ontology). The benefit of using MSDA Transformers is that they allow for the flexible integration of alternative methods for collecting service context information; as

new context APIs are defined and implemented in services, new MSDA Transformers can be deployed to support them.

### 4.3. Context Utilization

MSDA Managers and Bridges are the components that directly process and utilize available context information. MSDA Managers are responsible for selecting instances of services that meet user requirements. MSDA Bridges are responsible for taking the decision on whether or not to propagate a discovery request to a certain network domain. If a client does not specify any user context information, MSDA only processes network and service context information for controlling the dissemination of the client's request.

In the following, the example of Figure 3 is used to explain how context information is utilized in the different phases of the service discovery process. The user *Ratif* generates the discovery request (Step 1) in which he specifies context parameters and rules. Upon receiving this request, as no services are available in *Network 1*, *MSDA\_Manager<sub>1</sub>* forwards it to *MSDA\_Bridge<sub>12</sub>* (Step 2). To decide whether the request has to be propagated, and the destination network *Network2* can accept such request, *MSDA\_Bridge<sub>12</sub>* (Step 3) processes applicable network context rules associated with *Network1* and *Network2*, and user context rules included in the discovery request (i.e., network load should not be high, and network bandwidth should be medium/high). As all rules are positively matched, the request is forwarded to *MSDA\_Manager<sub>2</sub>*, which in turn forwards it to the local SDA Plugins for processing. Once *MSDA\_Manager<sub>2</sub>* has received the result from the SLP SDA plugin (i.e., the MSDA description for S2), it verifies the appropriate service and user context rules (Step 4) to ensure that the service matches the user requirements (i.e., secure and free of charge service) and that the user's device is able to fully support the service (i.e., medium/high-resolution screen).

	User context	Service context	Network context
<b>Carried in</b>	MSDA Request	MSDA Description	MSDA Context messages
<b>Provided by</b>	User	Service and Transformer	MSDA Manager
<b>Used by</b>	MSDA Manager and Bridges	MSDA Manager	MSDA Manager and Bridges
<b>Used for</b>	Request propagation and service selection	Service selection	Request propagation and service selection

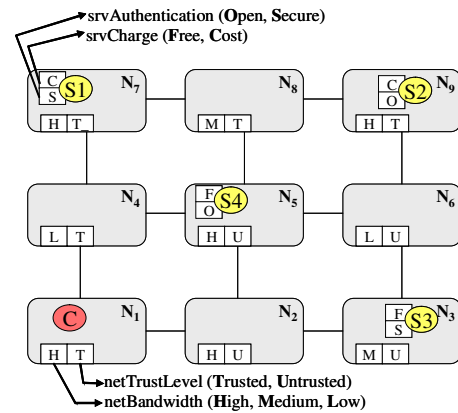
**Table 1: Context management and utilization**

To summarize, Table 1 describes which entities manage, create, and use the different types of context information (policies and parameters).

## 5. Implementation and Evaluation

We implemented a first prototype of MSDA in Java (available at [11]). XML has been used to specify context information and messages exchanged between MSDA components. JaxMe has been used to generate the Java bindings for XML messages. MSDA components communicate over multicast (context information and service discovery) and unicast (service access) sockets. The prototype integrates multiple SD protocols to experiment with both pull-based (e.g., Ariadne [12]) and push-based (e.g., SLP [7], UPnP [13]) protocols.

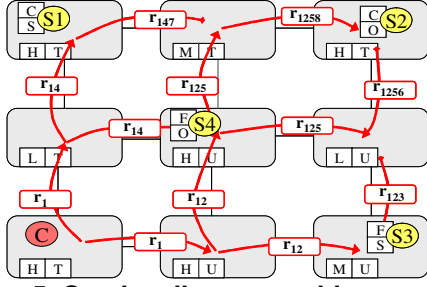
To assess in terms of performance the use of context information in MSDA, we consider a topology consisting of nine networks ( $N_1$ - $N_9$ ) depicted in Figure 4. Context information of each network includes two parameters: (i) available bandwidth, and (ii) trust level. One client *C* resides in  $N_1$ , while four video streaming services (S1 to S4) are in four different networks. Service context information specifies (i) cost and (ii) authentication method.



**Figure 4: Network topology**

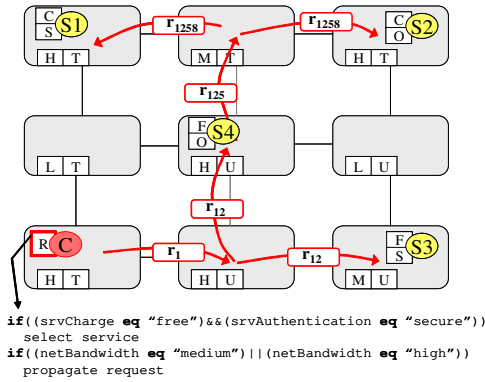
### 5.1. Assessing Context-awareness in MSDA

When no context information is used, an MSDA discovery request generated by *C* for video streaming services is propagated according to the paths displayed in Figure 5. For example, the request  $r_{1258}$  arriving in network  $N_9$  followed the path  $N_1$ - $N_2$ - $N_5$ - $N_8$ . Service responses are then returned according to the reverse propagation path of the request (e.g., the service description of S2 is returned following the path ( $N_9$ - $N_8$ - $N_5$ - $N_2$ - $N_1$ )). In total, 22 messages are exchanged and all 4 services are discovered.



**Figure 5: Service discovery without context**

However, client C may be interested only in services that are free of charge and secure, and with network connectivity good enough to support high-quality videos. With the context support of MSDA, these specific user requirements can be expressed by means of context rules carried in the discovery request. Figure 6 illustrates the propagation of such a context-aware MSDA discovery request. Compared to the context-less request, only 8 messages are exchanged and the client is provided only with S3.



**Figure 6: Context-aware service discovery**

Table 2 summarizes the number of messages and sets of service descriptions returned for service requests carrying different context rules. These examples demonstrate how the use of context information: (i) reduces the number of exchanged messages thus saving bandwidth and energy in resource-constrained networks; and (ii) enables the selection of services that meet the user's requirements and comply with the environment conditions.

context rules	number of messages	service results
No context rule	22	S1..S4
All context rules true	22	S1..S4
Medium/high netBandwidth	16	S1..S4
Medium/high netBandwidth AND free secure service	8	S3
TrustedNetwork AND free of charge secure service	6	S1

**Table 2: Multi-hop Context-aware SD**

## 5.2. Context Processing Overhead

We ran experiments<sup>3</sup> aimed at evaluating the overhead due to the processing of context information while performing service discovery.

First, we measured the service discovery (SD) time of a single service advertised in the local network while varying the number of context rules in the discovery request. Table 3 presents the SD times in ms in the case 0, 5, 10, 20, and 40 user context rules (CRs) are specified in the discovery request. In all tests, the service description has 40 context parameters that match the context rules. The discovery time increases between 1.0ms and 1.6ms for each added context rule, which respectively correspond to 2.1% and 3.4% of the context-less SD time.

	0 CR	5 CR	10 CR	20 CR	40 CR
SD time (ms)	47	55.5	63	75.5	88.5

**Table 3: Local Context-aware SD**

These SD times include both the communication times between client and MSDA Manager (i.e., time for sending the discovery request to the MSDA Manager and time for sending the result to the client) and the processing time of the discovery request. Based on previous experiments [11], the network communication time (i.e., over Ethernet, Bluetooth, or WiFi) is negligible compared to the message processing time that includes data marshalling and unmarshalling times.

To more specifically estimate the actual overhead due to the processing of context information, we also measured marshalling and unmarshalling times of discovery requests carrying different combinations of context information. Table 4 shows the processing time for marshalling and unmarshalling a discovery request with 0, 5, 10, 20, and 40 context rules (CRs). Table 5 presents the total processing time for marshalling and unmarshalling a discovery request with 0, 10, 20, 50, and 100 context parameters (CPs). The total time for marshalling and unmarshalling a context-less discovery request is about 4.6ms. For a context-aware discovery request this time increases between 0.2ms and 0.4ms for each added context rule and between 0.09ms and 0.14ms for each added context attribute. From these results, we can calculate that, out of the extra 8.5ms required for the processing of the discovery request with 5 CR and 40 context parameters, about 7ms is added by the marshalling and unmarshalling of the

<sup>3</sup> All tests were performed on a PC with a Pentium IV rated at 2.6 GHz. The service was advertised using UPnP. For each test we conducted 2 sets of 50 measurements, removing each time the two lowest and highest values, and calculated median value of the 2 sets.

context information, and therefore 1.5ms is due to the processing of the 5 context rules.

	0 CR	5 CR	10 CR	20 CR	40 CR
Marshall	1.2	1.9	2.7	3.8	4.7
Unmarshall	3.4	4.7	5.5	6	9.3

**Table 4: Context rules overhead (ms)**

	0 CP	10 CP	20 CP	50 CP	100 CP
Marshall	1.2	1.6	2.9	4.5	6.4
Unmarshall	3.4	3.9	4.5	6.4	7.3

**Table 5: Context parameters overhead (ms)**

As shown in Table 2, even a limited number of context information can greatly improve the performance of the service discovery and bring notable benefits to the user. Our experiments have shown that the additional overhead due to context information is limited compared to a context-less SD time. We expect typical context information (user, service, or network) to contain about 5 to 10 context rules and 20 to 40 context parameters.

## 6. Conclusions and Future Work

The MSDA middleware platform has been designed to support service discovery in heterogeneous and multi-protocol pervasive environments. We further built and integrated in MSDA the support for context-awareness in order to accomplish more efficient service discovery. The key feature of our approach is that context information is used not only to select the most appropriate service instance, but also to improve the dissemination of service requests across heterogeneous pervasive environments, thus minimizing the resource consumption. Moreover, the use of Transformers facilitates the collection and integration of context information from services advertised with traditional, context-less SD protocols. As demonstrated by the evaluation, properly managing context information not only produces more accurate results, but also reduces the number of exchanged messages, at the cost of a limited overhead on the whole service discovery time.

In future developments of MSDA, we plan to investigate how to support proactive provisioning of context information, how to optimize the exchange of such information among MSDA components (e.g., by aggregating network context information exchanged between MSDA Bridges), and how to guarantee the formulation of correct and non-conflicting context rules. Furthermore, we will investigate how to specialize the support offered by existing context ontologies to accomplish both efficient service selection and discovery request propagation.

## 7. References

- [1] L. Capra, S. Zachariadis, and C. Mascolo, "Q-CAD: QoS and Context Aware Discovery Framework for Mobile Systems", In *Proc. of Int'l Conference on Pervasive Services (ICPS'05)*, Greece, July 2005.
- [2] G. Chen, and D. Kotz, "Solar: An Open Platform for Context-aware Mobile Applications", In *Proc. of the 1<sup>st</sup> Int'l Conference on Pervasive Computing (Pervasive 2002)*, Switzerland, June 2002.
- [3] H. Chen, T. Finin, and A. Joshi, "The SOUPA Ontology for Pervasive Computing", Whitestein Series in Software Agent Technologies. Springer, July 2005.
- [4] A. K. Dey, D. Salber, and G.D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications". *Human-Computer Interaction*, Vol. 16 (2-4), pp. 97–166, 2001.
- [5] Foundation for Intelligent Physical Agents. FIPA Device Ontology Specification. Geneva, Switzerland. 2002. Specification number SI00091.
- [6] Foundation for Intelligent Physical Agents. FIPA Quality of Service Ontology Specification. Geneva, Switzerland. 2002. Specification number SC00094.
- [7] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2", IETF RFC 2608, June 1999 (<http://www.ietf.org/rfc/rfc2165.txt>).
- [8] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran, "Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies". W3C Recommendation, January 2004.
- [9] C. Lee and S. Helal, "Context Attributes: An Approach to Enable Context-awareness for Service Discovery", In *Proc. of the 2003 Symposium on Applications and the Internet (SAINT'03)*, Orlando, USA, January 2003.
- [10] G. Lee, P. Faratin, S. Bauer, and J. Wroclawski, "A User-Guided Cognitive Agent for Network Service Selection in Pervasive Computing Environments", In *Proc. of the 2<sup>nd</sup> IEEE Int'l Conference on Pervasive Computing and Communications (PerCom'04)*, Orlando, USA, March 2004.
- [11] MUSDAC Middleware Platform: <http://www-rocq.inria.fr/arles/download/ubisec/index.html>.
- [12] F. Sailhan and V. Issarny, "Scalable Service Discovery for MANET", In *Proc. of the 3rd IEEE Int'l Conference on Pervasive Computing and Communications (PerCom'05)*, Kauai Island, USA, March 2005.
- [13] UPnP™ Forum, Universal Plug and Play: <http://www.upnp.org/>.
- [14] F. Zhu, M. Mutka, and L. Ni, "Splendor: A Secure, Private, and Location-aware Service Discovery Protocol Supporting Mobile Services", In *Proc. of the 1st IEEE Int'l Conference on Pervasive Computing and Communications (PerCom'03)*, Fort Worth, Texas, USA, March 2003.