Interoperability of Service Discovery Protocols: Transparent versus Explicit Approaches

Yerom-David Bromberg, Valérie Issarny, Pierre-Guillaume Raverdy
INRIA-Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay, France,
firstname.lastname@inria.fr

***Abstract*— Discovering networked services in pervasive computing environments is problematic as multiple Service Discovery Protocols (SDPs), differing on their services description formats as well as advertisement and request models, have already become (de-facto) standards. This calls for a solution to SDP interoperability, enabling clients to locate networked services independent of the SDP they use to advertise their presence. In this paper, we report on our experience in developing two such solutions: the transparent approach of INDISS based on message translation, and the explicit approach of MSDA based on protocol integration. While efficient and able to support legacy clients and services, INDISS is limited by the basic service information available in existing SDPs, and assumptions about the network protocols used by the SDPs. Advanced discovery features required by pervasive environments, such as context or security management, can only be provided by more complex discovery frameworks like MSDA, but come at a price.**

***Index Terms*—Service discovery, interoperability, pervasive environments**

I. INTRODUCTION

The availability of consumer-oriented mobile devices powerful enough to host services, and the deployment of heterogeneous networks based on wireless networking technologies have enabled the emergence of service-rich computational environments aimed at supporting users in their daily life. Discovering the networked services available in such environments is a crucial first step for providing a satisfying experience to the user.

Over the years, many academic and industry-supported *Service Discovery Protocols* (SDPs) have been proposed (e.g., Jini, SSDP, SLP, and UDDI). Maheswaran [3] proposes a taxonomy of discovery protocols based on their organization (i.e., centralized vs. decentralized vs. hierarchical), methods (push vs. pull vs. symmetric), and timing (irregular vs. periodic). One common characteristic of these SDPs is that they have been designed based on specific assumptions about the underlying network (e.g., Internet, home network), the users' behavior, or the applications' needs. While efficient for the targeted environment, they prove inefficient or not applicable in different settings (e.g., network-flooding [4]). Another characteristic is that they do not directly interoperate with each other's as they employ incompatible formats and protocols for service description, service advertisements, or discovery requests. Furthermore, these SDPs are often integrated in middleware platforms (e.g., SSDP and UPnP), complicating interoperability (i.e., incompatible data types or communication models). In fact, the diverse environment constraints and the de-facto standard status of some of the existing SDPs, make it unlikely for a new and unique SDP to emerge. Several projects are thus investigating interoperability solutions for SDP [5][6][7][8], as requiring clients and services to support multiple SDPs is not realistic.

SDP interoperability is typically achieved using intermediate representations of SD paradigms (e.g., service description, discovery request) [5][6] instead of direct mappings [8], as the latter does not scale well with the number of supported SDPs. Two approaches are however possible to interface with the various SDPs: transparent or explicit. In the transparent approach, the interoperability layer is located close to the network and directly translates SDPs messages to/from the various SDPs [8]. Clients and services are unaware of the translation process. In the explicit approach, the interoperability layer is located on top of the existing SDPs, and provides an explicit discovery API to clients (and sometimes services) [5]. While the transparent approach eases the deployment and use of the interoperability solution by legacy clients and services, the explicit approach enables the extension of existing SDPs with advanced features such as context management [8].

In this paper, we detail and compare the SDP interoperability solutions developed in the IST Amigo[1] (Section II) and IST UBISEC[2] (Section III) projects that are respectively based on the transparent and explicit approaches. The comparison of the two approaches (Section IV) highlights the different networking environments targeted by the two solutions as well as the different methodology for supporting client/service interactions in pervasive environments. We conclude (Section V) by outlining on-going work in both projects as well as in the IST Plastic project[3] to support service access in the heterogeneous computing environment, and discussing partial integration of the two solutions.

This work described herein is funded by the European Commission under the FP6 IST project Amigo contract number 004182, the FP6 IST project UBISEC contract number 506926, and the FP6 IST Project Plastic contract number 026955.

[1] IST Amigo project: http://www.amigo-project.org/
[2] IST UBISEC project: http://www.ubisec.org/
[3] IST Plastic project: http://www.ist-plastic.org/

II. SERVICE DISCOVERY IN AMIGO

The IST Amigo project aims to improve the usability of home networks, and for that investigates: (i) interoperability issues arising from the use of heterogeneous networking technologies, devices, middleware platforms, and standards, (ii) the automatic discovery of devices and services as well as their composability and upgradeability and self-administration, and (iii) intelligent user interfaces. In the context of the Amigo project, we designed INDISS [1] (Interoperable Discovery System for Networked Services) to overcome SDPs heterogeneity. In this section, we detail the networking environment and the design goals of INDISS leading to the use of the transparent approach. We then present the main components and deployment of the system, and discuss how SDPs messages are translated into sets of events exchanged between the system's components. We then emphasize how the transparent approach enables support for legacy applications and helps future developments.

A. Amigo networking model

Amigo assumes an all-IP networking environment, meaning that clients and services are using communication protocols built on top of IP and that all devices within the pervasive environment (i.e., home network) are IP reachable. While different devices may belong to different subnets, unicast and multicast packets can reach all devices. Due to the environment constraints, SDPs in home environment are multicast based and fully distributed.

B. A transparent approach

Keeping with the goal of usability, and the focus on dynamic home networks, INDISS is specifically designed as a transparent solution minimizing resource usage (i.e., memory, processing and bandwidth), and introduces lightweight mechanisms that may be adapted easily to any platform. The transparent approach is also key to support legacy clients and services, which is crucial in Amigo. INDISS is composed of a set of event-based components and their composition is performed dynamically at run-time according to both the context and the device on which INDISS is deployed. INDISS operates close to the network, capturing and translating network messages without the need for clients or services interactions. As a result, service discovery interoperability is provided to applications without altering them: applications are not aware of the existence of INDISS.

C. INDISS architecture

INDISS main components are the *monitor*, the *parser*, and the *composer*:

- The *monitor component* detects the SDPs that are used based on network activity on the assigned multicast groups and ports. This component also captures/collects network messages sent by clients and services onto these multicast groups, and forwards them to the appropriate parser components.
- The *parser* component, associated to a specific SDP, transforms the raw data flow (i.e., network messages) into series of events, extracting semantic SDP concepts from syntactic details of the SDP messages. The generated events are delivered to an *event bus* locally deployed.
- The *composer* component delivers a SDP message understood by the target clients and/or services based on specific sets of events received from the *event bus*.

Parsers and composers are dedicated to specific SDP protocols. In INDISS, the communication between the *parser* and the *composer* does not depend on any syntactic detail of any protocol. They communicate at a semantic level through the use of events. A fixed set of common events has been identified for all SDPs, and each SDP has also a set of specific events. For example, a subset of events generated by a UPnP parser are successfully understood by a SLP *composer,* whereas specific UPnP events, due to UPnP functionalities that SLP does not provide, are simply discarded from the SLP *composer,* as they are unknown.

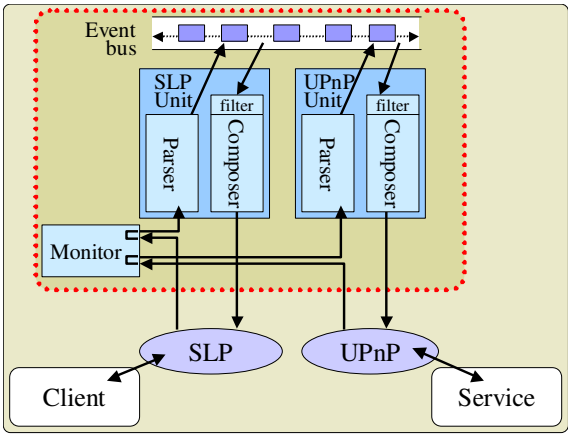

Figure 1: INDISS Architecture

SDP interoperability comes from the composition of multiple parsers and composers dedicated to different SDPs, and the implicit creation of an event bus. As depicted in Figure 1, the monitor receives a incoming UPnP message from the service, and forwards it to the parser of the UPnP unit. The UPnP parser then translates the message into a set of events. The SLP composer receives the relevant (subset of) events according to its event filters, and composes the adequate SLP messages. This message is then sent onto the SLP multicast group and received by the client.

In general, SDP functions like service request, service registration or service advertisements, are complex distributed processes that require coordination between the actors of the specific service discovery function. It follows that the translation of SDP functions that is realized by INDISS is actually achieved in terms of translation of processes and not simply of exchanged messages, further requiring coordination between the parser and composer. This is realized by embedding the parser and composer within a *unit* that runs coordination processes associated with the functions of the supported SDP. Specifically, the behavior of the unit is specified using finite state machines [1].

D. INDISS deployment

INDISS may be deployed independently on one or more devices in the network. Each instance of INDISS dynamically

instantiates parsers and composers for the SDPs it supports, and provides interoperability between these SDPs for all the devices in the network. Multiple instances of INDISS may provide interoperability between different sets of SDPs. While INDISS can be deployed on a home gateway, and translate and forward all messages, it may take into account the CE nature of home networks, where some devices only provide services, while others only accesses them. Indeed, INDISS can be optimized for client-only or service-only devices by reducing the number of messages sent.

E. *Unique feature: transparent interoperability*

With INDISS, application components continue to use their own native service discovery protocol; interoperability is achieved through a transparent integration of INDISS. Event streams are totally hidden to components outside INDISS, as they are assembled into specific SDP messages through *composers*. Consequently, interoperability is guaranteed to existing applications tied to a specific SDP without requiring any change to applications. Similarly, future applications do not need to be developed with a specific middleware API to benefit from SDP interoperability.

III. SERVICE DISCOVERY IN UBISEC

The IST UBISEC project investigates the discovery, security, and customization of services in Beyond 3G (B3G) networks. As UBISEC targets a highly heterogeneous networking environment, the *Multi-protocols Service Discovery and Access* (MSDA) platform [2] was designed to provide context-based enhanced service discovery. In this section, we detail the networking environment and the design goals of MSDA that lead to the use of the explicit approach. We then present the components of the platform, the intermediate representation of service descriptions, and the deployment of the platform in a multi-networks environment. We finally detail how context information is used both for filtering results, but also to control the dissemination of service information.

A. *UBISEC networking model*

UBISEC also considers an all-IP networking environment. However, UBISEC assumes highly heterogeneous, loosely connected networks. Network heterogeneity arises from: (i) the use of different networking technologies, (ii) variations in the sets of IP-level configuration and communication functionality provided by the network (e.g., IP multicast support, DHCP), and (iii) networks belonging to different administrative domains (e.g., public vs. restricted) and management models (infrastructure-based vs. ad hoc). These networks may further target different classes of applications (e.g., sensor networks, home/automation networks, hotspots). One major consequence is that we consider that global IP routing is not guaranteed. In UBISEC, the pervasive environment is modeled as a dynamic composition of heterogeneous networks. Moreover, the Internet and cellular networks are considered as conduits enabling the interconnection of the various networks.

B. *An explicit approach*

MSDA is an additional layer on top of existing SDPs. MSDA is instantiated independently in each network of the environment, and each instance registers as a service (the *MSDA Service*) with the active SDPs in the network. The MSDA Service provides a pull-based service discovery interface (i.e., clients issue discovery requests and are returned the matching services). MSDA-aware client applications interact with services that use different discovery and access protocols, or are in different networks, explicitly through MSDA (i.e., using the format and protocols of MSDA). MSDA instances in nearby networks dynamically communicate with each other to disseminate service information and to provide remote service access. Each MSDA instance independently selects with which nearby MSDA instance to connect to, and filters service information and access requests that it receives/sends from/to these MSDA instances.

C. *MSDA architecture*

MSDA main components are (See Figure 2):

- The *MSDA Manager* that manages discovery and access requests within the network for local and remote clients,
- *Plugins* that interact with specific SDPs to collect service information, register the *MSDA service* to be used by local clients, and perform service access on behalf of remote clients,
- *Transformers* that extend service descriptions with context information,
- *MSDA Bridges* that assist MSDA Managers in expanding the service discovery and service access to other networks in the whole pervasive environment.

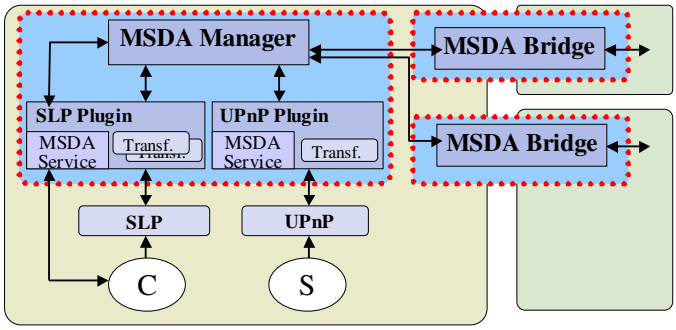

Figure 2: MSDA Architecture

D. *MSDA deployment*

MSDA can be deployed on one or more devices in a network (one instance per device), and one of the MSDA instances is dynamically elected as the MSDA Manager of the network. We implemented the MSDA Manager as a centralized component, as the size of a network (e.g., subnet) is limited by nature (i.e., its number of devices and services is limited). The MSDA Manager is the main component of MSDA, as it processes, within its network, discovery and access requests from local and remote clients. The MSDA Manager periodically sends presence beacons so that other MSDA-aware devices in the network can detect its absence, as well as duplicates, and recover. The elected MSDA Manager then activates its SDA Plugins and transformers. The MSDA Manager may dynamically select some of the MSDA instances in its network to act as MSDA Bridges (if they registered as

potential MSDA Bridge) based on several criteria (i.e., connectivity to other networks, expected lifetime, processing power, and cost). An MSDA Bridge disseminates discovery requests between the networks on its different network interfaces. In Figure 2, 3 instances of MSDA have been started in the network (one MSDA Manager and two MSDA Bridges). The MSDA Manager has started an SLP and a UPnP plugin. The MSDA-aware SLP client discovers the MSDA Service with SLP, and accesses it to discover the services advertised in UPnP as well as services in remote networks.

E. MSDA description and request

In MSDA, services are described using the *MSDA Description* format, which is a generic and modular service description format. In addition to the service information collected from the SDP-specific service description, the MSDA description also contains information to assist the remote access to the service (e.g., network path leading to the destination network, access protocols supported) and information to control the dissemination of the description (e.g., minimum bandwidth requirements). An SDA Plugin first generates an initial MSDA Description based on the SD-specific service description it receives, and forwards it to its Transformers that will extend the MSDA description with service-specific context information. The MSDA Description is then forwarded to the MSDA Manager that adds network-specific context information. Each MSDA Bridge that forwards a description also extends it with context and propagation information. Discovery requests in MSDA are created by MSDA Managers on behalf of client applications and are similar to service descriptions. In addition to the information describing the requested service, the discovery request also contains context information to constrain the dissemination of the request and filter the services returned by the SDPs as well as *request processing* information that defines how MSDA Managers, both local and remote, should return the result of the MSDA Request (e.g., timeout, partial results).

F. Unique feature: context-aware service discovery

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. Context information is stored in MSDA descriptions and requests, and is processed by MSDA Bridges to control the dissemination of MSDA messages, and by MSDA Managers to filter the results of a request.

IV. COMPARISON

Based on the descriptions in Sections II and III, we summarize the two SDP interoperability solutions proposed by the Amigo and UBISEC projects in Table 1.

INDISS/Amigo	**MSDA/UBISEC**
Networking environment	
Home networks - heterogeneous networks within same administrative domain	B3G Networks - dynamic composition of independent networks
Interoperability requirements	
Lightweight discovery Applicable to CE devices No infrastructure building	Enhanced discovery (context, security) Multi-networks discovery
Approach to SD interoperability	
Translation of SDPs messages - transparent discovery - network messages interception - event-based parsing and composition	Integration of SDPs - explicit API - enhanced service description - client-side interoperability
Strengths	
Transparent to clients & services Limited resources requirements Easy deployment	Context-aware discovery Multi-network support Controlled dissemination
Weakness	
Limited network reach Intersection of SD information	Requires client support High processing requirements

Table 1: INDISS and MSDA Comparison

The comparison of the two approaches first highlights that the underlying networking environment does not directly intervene in SDP interoperability (i.e., the conversion of SDP paradigms such as service descriptions), it influences the sets of SDPs targeted and the required features of the interoperability layer. The comparison also highlights the difference between the two approaches is terms of service information conveyed in the intermediate representation and its generation process.

A. Networking model

Amigo and UBISEC differ on their networking model for pervasive environments, with the interconnection of different networks being managed either at the network level (INDISS), or at the application level (MSDA). The Amigo model is appropriate when considering a single administrative domain (e.g., home environment) but requires global IP routing as well as advanced routing and filtering management solutions in a B3G environment. As this is unlikely (in the short or mid-term), an MSDA-like bridging extension to INDISS would be required for such environment.

INDISS and MSDA also differ on their deployment models. MSDA requires a strong coordination between its different components (within a network and between neighboring networks). This network overhead is however limited and independent of the number of clients and services. INDISS on the other hand does not provide/ any coordination between multiple INDISS instances and therefore does not generate any control overhead. However, as multiple instances of INDISS in a network operate independently, it may create unnecessary duplication of messages as requests and/or advertisements are translated and reemitted by each instance. We currently investigate a lightweight coordination protocol between INDISS instances to reduce such duplication

Network reconfiguration (split/merge) does not affect INDISS as no coordination exists between multiple instances of INDISS. However, a network split may discontinue INDISS support for some devices, and a network merge may introduce message duplication. MSDA detects network reconfigurations and new MSDA Managers and Bridges may be elected following a split/merge, creating a temporary network overhead and potentially canceling ongoing MSDA requests and service access

B. *Intermediate representation*

The transparent versus explicit approach to SDP interoperability affects the amount of service information used/available. In INDISS, the service information carried over when translating between two SDP corresponds to the intersection of the two sets of service information. In MSDA, as an explicit API is used by clients applications, we designed the MSDA description as an extensible format containing the union of the information contained in the service description formats of existing SDPs, enriched with context information. As more services are deployed in highly heterogeneous environments, selecting the most effective service instance based on context information will be crucial.

INDISS and MSDA also differ on how the reconfiguration of the translation process from SDP specific messages to the solution's intermediate representation (INDISS events and MSDA description/request). In MSDA, a new SD plugin (Java class) must be dynamically loaded and instantiated, while in INDISS, the Final State Machine (FSM) that performs the conversion of SPD messages can be dynamically reconfigured at run-time. INDISS approach is therefore more efficient in case of frequent fine-grain updates. More importantly, INDISS approach benefits from the numerous tools available to optimize and validate FSMs.

C. *Performance evaluation*

Prototypes have been implemented for both solutions and performance results clearly spell out the reactivity of INDISS compared to MSDA. Indeed, discovering an SLP service takes around 40 ms for a UPnP client with INDISS (1.2 ms when INDISS is deployed on the client), while it takes about 320 ms with MSDA. Most of the overhead is however related to the SOAP processing of the service call (the same discovery using a socket-based implementation of the MSDA service takes about 105 ms).

V. CONCLUSION

Performance evaluation highlighted the high cost of service discovery in MSDA compared to existing SDPs and INDISS. This overhead is directly related to the service API of MSDA, which is required for any enhanced service discovery protocol. For local interactions (i.e., within the same administrative domains such as an intranet or home network), the INDISS approach is therefore more appropriate as it supports responsive applications. We are evaluating the combination of the two approaches in the context of the IST Plastic project. A potential integration is to use INDISS for translating service announcements and discovery requests between SD specific format and the MSDA format (i.e., replacing the SDA plugins).

Most SDP interoperability projects do not investigate the interoperability of service access protocols [5][8]. However, addressing this issue is crucial as discovering services that cannot be accessed is inconsistent/incoherent. Interoperability of service access protocols either relies on additional code [6] or reflection support [7]. We are now investigating service access for both projects. In Amigo, we reuse event-based parsing techniques to design NEMESIS as an independent layer providing service access interoperability. For the MSDA platform was currently focus on supporting service access in multi-networks configuration.

The INDISS/MSDA comparison also highlights the differences in the Amigo and UBISEC approaches for supporting interaction between clients and services. While UBISEC puts forward a integrated platform for context-aware service discovery and access in multi-networks environment, Amigo provides independent solutions for service discovery (INDISS), service access (NEMESIS), or context management. The UBISEC approach enables a better integration of the different facets (discovery, access, context) in its components allowing, for example, the use of context information to limit the dissemination of service information. Amigo however promotes reusability, supports a better distribution of the time-consuming tasks to the appropriate devices in the network, and better supports updates of existing components.

REFERENCES

[1] Yerom-David Bromberg, Valerie Issarny. "INDISS: Interoperable Discovery System for Networked Services". In *Proceedings of ACM/IFIP/USENIX 6th International Middleware Conference (Middleware).* 2005.

[2] Pierre-Guillaume Raverdy *et al*, "Efficient Context-aware Service Discovery in Multi-Protocol Pervasive Environments", In *Proceedings of IEEE International Conference on Mobile Data Management (MDM), 2006.*

[3] M. Maheswaran, "Data Dissemination Approaches for Performance Discovery in Grid Computing Systems", In Proceedings of 15th International Parallel and Distributed Processing Symposium, 2001.

[4] S.Y. Ni *et al*, "The Broadcast Storm Problem in a Mobile Ad Hoc Network", In Proceedings of the ACM/IEEE 5th International Conference on Mobile Computing and Networking(MobiCom), 1999.

[5] Adrian Friday *et al*. "Supporting service discovery, querying and interaction in ubiquitous computing environments". *ACM Baltzer Wireless Networks (WINET) Special Issue on Pervasive Computing and Communications*, 10(6):631-641, 2004.

[6] J. Allard *et al*, "Jini Meets UPnP: An Architecture for Jini/UPnP Interoperability," In Proceedings of the International Symposium on Applications and the Internet (SAINT), 2003.

[7] P. Grace *et al*, "ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability". In Proceedings of International Symposium on Distributed Objects and Applications, 2003.

[8] T. Koponen *et al*, "Service Discovery: A Service Broker Approach" In Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS), 2004.

[9] C. Lee, S. Helal, "Context Attributes: An Approach to Enable Context-awareness for Service Discovery" in 2003 Symposium on Applications and the Internet , 2003