Service Discovery in Ubiquitous Feedback Control Loops

Daniel Romero, Romain Rouvoy, Lionel Seinturier, Pierre Carton

Service Discovery in Ubiquitous Feedback Control Loops

Daniel Romero, Romain Rouvoy, Lionel Seinturier, and Pierre Carton

INRIA Lille-Nord Europe, ADAM Project-team
University of Lille 1, LIFL CNRS UMR 8022
59650 Villeneuve d'Ascq, France
firstname.lastname@inria.fr

Abstract. Nowadays, context-aware applications can discovery and interact with services in ubiquitous environments in order to customize their behavior. In general, these providers use diverse discovery and interaction protocols. Furthermore, they can join and leave the environment at anytime, making difficult the utilization of services. Therefore, this mobility and variability in terms of protocols impose a low coupling between the interacting entities and the need for spontaneous communications. Unfortunately, the existing works in literature fail to deal with these needs in a simple and flexible way. In this paper we face this problem by defining *ubiquitous bindings* for SCA (Service Component Architecture) applications. These bindings modularize the discovery concerns promoting sharing of common discovery functionalities and simplifying the integration of discovery protocols. In this way, these bindings enable the transparent advertisement, discovery, filter, and access of services available in the environment. Our ubiquitous bindings are integrated into the FraSCAti platform and their benefits are demonstrated by building ubiquitous feedback control loops.

1 Introduction

In ubiquitous environments, different computational entities (mobile devices, laptops, sensors, etc.), both providing and consuming services, arrive and leave routinely [1]. The different applications executing in these environments exploit this richness to adapt and improve their behavior. However, in ubiquitous environments, because there is no standard discovery protocol, the service providers are used to select the most suitable according to their capabilities. This results in a collection of advertised services using heterogeneous discovery and interaction protocols, and characterized by their dynamism, that cannot always be discovered and accessed by clients. Therefore, spontaneous communications become an important issue to deal with dynamicity and unpredictability of ubiquitous environments [1]. Unfortunately, the works in the literature to deal with this issue tend to be complex, not flexible enough or do not consider the needs of consumers in terms of interaction mechanism [2,3,4].

In this paper, we extend the SCA (Service Component Architecture) model [5] with the notion of ubiquitous bindings in order to support service

discovery in ubiquitous environments. These bindings enable applications to discover and filter SCA services available in the environment, which are then accessed using the supported SCA bindings. We claim that SCA extensibility and independence from communication and implementation technologies allow the transparent management of service discovery. Furthermore, by encapsulating spontaneous communications in SCA bindings, we provide the flexibility required for choosing the most suitable discovery and communication mechanism between consumers and providers. We illustrate the utilization of our bindings by designing *ubiquitous* feedback control loops (FCLs) [6]. This kind of FCLs allows us to deal with runtime adaption of context-aware applications supporting the mobility of the participating entities. The ubiquitous bindings are implemented as SCA components for promoting reuse and integration into the FraSCAti platform [7].

The rest of this paper is organized as follows. We start by introducing the motivations for discovery and adaptation in the landscape of context-aware applications (cf. section 2). We continue with the foundation of our proposal (cf. section 3) before introducing our ubiquitous FCLs (cf. section 4). Then, we present our lightweight solution for supporting discovery in SCA applications (cf. section 5) and the evaluation of our implementation (cf. section 6). In section 7, we compare our approach with existing solutions for service discovery in ubiquitous environments. Finally, we summarize the conclusions of this work and the promising research directions in section 8.

2 Motivations and Challenges

This section highlights the challenges for context-aware applications in ubiquitous environments. We start by describing a motivating scenario (cf. section 2.1) before introducing the challenges it exhibits (cf. section 2.2).

2.1 Motivating Scenario

A smart home generally refers to a house environment equipped with various types of sensor nodes, which collect information about the current temperature, occupancy (movement detection), noise level, and light states. In addition to that, actuators are also deployed within rooms to physically control appliances, such as lights, air conditioning, blinds, television, and stereo. In this environment, both sensors and actuators can be accessed from mobile devices owned by the family. Furthermore, the control system deployed in such a smart home is able to retrieve preferences about room configuration from mobile devices and change the room state according to them. For example, preferences can describe the temperature and light levels accepted by each family member. When several people share the same room, the decision is based on merged preferences. In case of conflict, the decision prioritizes the first person that enters the room. The mobile devices also have an application that enables users to control the appliances in home. This application has several modules (one for each appliance)

that are activated or deactivated according to the current *battery level*, the *battery saving* preference and *activation of modules* preference (this information is also provided by the mobile device). The modules also are installed/uninstalled regarding the changes in the appliance configurations. The following paragraphs describe two concrete situations of the scenario.

Alice listens to music in the living room. The temperature conforms to her preferences. When Bob enters the living room, the controller detects his device and retrieves the preferences (related to the room configuration as well as for the application allowing the access to appliances). Let assume that there is no conflict with the temperature level. However, the light level is too low for Bob's tolerance. The light range specified by Alice includes the level accepted by Bob. Hence, the system decides to modify the light level of the room according to Bob's preferences. On the other hand, the system analyses battery level and decides that it must deactivate the multimedia module allowing downloads from a multimedia server available at home.

In another situation, Alice installs a new TV. The system detects the new device and retrieves the required module to control it. When Bob arrives home, the system detects his device and installs the module as well.

2.2 Key Challenges

According to our scenario, we can identify three key challenges for context-aware applications in ubiquitous environments: *i) heterogeneity*, *ii) mobility* and *iii) runtime adaptation*. The first one refers to the variability in terms of devices (that differ in their processing capabilities), services (implemented with several technologies) and context information (that has different syntax and semantic) present in the environment. This requires a flexible solution in terms of communication (*e.g.*, protocols and data representation) that allows applications to access the context and services. The mobility is concerned with the dynamicity of services and context providers, which can spontaneously join and leave (*e.g.*, mobile devices in the scenario). Hence, the applications should keep working even if some providers are gone as well as they should have the possibility to discover new services. Finally, the adaptation of context-aware applications requires the retrieval and processing of the context information for deciding the needed reconfigurations. In this adaptation should be considered the variable capabilities of the different devices that execute the context-aware applications.

As presented in this paper, we face the mobility and heterogeneity issues using our RESTful (cf. section 3.3) and ubiquitous bindings (cf. section 5). We also provide some highlights for dealing with the adaptation at runtime by defining ubiquitous feedback control loops (cf. section 4).

3 Background

In this section, we present the foundation of our proposal—*i.e.*, the feedback control loops (cf. section 3.1) and SCA (cf. section 3.2). We also introduce the

FraSCAti platform (cf. section 3.2) and RESTful bindings (cf. section 3.3), which we use for implementing ubiquitous FCLs.

3.1 Feedback Control Loops for Autonomic Computing

The Autonomic computing enables the development of applications that exhibit properties such as *self-configuration*, *self-optimization*, *self-healing* and *self-optimization* [6,8]. These properties are generally achieved by means of the MAPE-K model, which is composed by the followings phases: *i) Monitoring* phase to collect, aggregate, and filter events from a managed resource, *ii) Analysis* phase that consist in the processing of the information recollected in the previous step, *iii) Planning* phase that defines the actions needed to achieve goals and objectives determined in the analysis, and *iv) Execution* phase for executing the plan determined in the previous step and using the adaptability capabilities of the system. All the different phases share the *Knowledge base* that includes historical logs, configuration information, metrics and policies.

3.2 The Service Component Architecture (SCA)

The Service Component Architecture [5] is a set of specifications for building distributed application based on SOA and *Component-Based Software Engineering* (CBSE) principles. In SCA, the basic construction blocks are the *software components*, which have *services* (or provided interfaces), *references* (or required interfaces) and expose properties. The references and services are connected by means of *wires*. SCA specifies a hierarchical component model. This means that components can be implemented either by primitive language entities or by subcomponents. In the latter case the components are called *composites*.

SCA is designed to be independent from programming languages, *Interface Definition Languages* (IDLs), communication protocols and non-functional properties. In this way an SCA-based application can be built, for example, using components in Java, PHP and COBOL. Furthermore, several IDLs are supported, such as WSDL and Java Interfaces. In order to support interaction via different communication protocols, SCA provides the notion of *binding*. For SCA references, *bindings* describe the access mechanism used to call a service. In the case of services, the bindings describe the access mechanism that clients have to use to invoke the service. Finally, an SCA component may be associated with *policy sets* or *intents* that declare the set of non-functional services that it depends upon. The SCA specification includes security and transactions policies [9], but the model may be extended with new ones if required.

The FraSCAti Platform. This platform [7] allows the development and execution of SCA based distributed applications. The platform itself is built as an SCA application, *i.e.*, its different subsystems are implemented as SCA components. FraSCAti extends the SCA component model to add reflective capabilities in

the application level as well as in the platform. Furthermore, the platform applies interception techniques for extending SCA components with non-functional services, such as confidentiality, integrity and authentication.

3.3 REpresentational State Transfer (REST)

REST [10] is an architectural style to define distributed applications. Typically, REST defines the principles for encoding (*content types*), addressing (*nouns*) and accessing (*verbs*) resources using Internet standards (*e.g.*, URIs, HTTP, XML and mime-types). Resources, which are key to REST, are addressable using a universal syntax (*e.g.*, a URL in HTTP) and share a uniform interface for the transfer of application states between client and server (*e.g.*, `GET/POST/PUT/DELETE` in HTTP). REST resources may typically exhibit multiple typed representations using—for example—XML, JSON, YAML, or plain text documents. Therefore, the simplicity, lightness, reusability, extensibility and flexibility properties that characterized REST make it a suitable option for exchanging context information in ubiquitous environments.

RESTful Bindings The REST bindings [11] follow the REpresentational State Transfer Principles. In this way, these bindings support multiple context representations (*e.g.*, XML, JSON and Java Object Serialization) and communication protocols (HTTP, XMPP, FTP, etc.). This flexibility allows us to deal with the heterogeneous context managers and context-aware applications as well as with the different capabilities of the devices that execute them. Details about the architecture of these bindings are presented in [11].

Synthesis : SCA provides a flexible and extensible component model that can be used in ubiquitous environments to deal with heterogeneity and mobility. In particular, as we present in section 5, we benefit from the protocol independence for defining ubiquitous bindings that enable spontaneous communications. Furthermore, the FraSCAti capabilities in terms of runtime adaptation for applications and the platform itself make it a good option for dealing with autonomic FCL in ubiquitous environments.

4 Ubiquitous Feedback Control Loops

In order to face the adaptation challenge in ubiquitous environments, we propose the architecture presented in Figure 1 to implement our ubiquitous FCLs [11]. We choose FCLs to support dynamic reconfigurations because they provide a clear isolation of the different steps of the adaptation process. This feature allows us to distribute the concerns in several entities and reduce the coupling between them. We give the "ubiquitous" adjective to these FCLs because they have the capacity to configure themselves at execution time. This means that some parts of the loop can dynamically join and leave, such as the applications running on mobile

devices in our smart home scenario. Furthermore, the low coupling between the FCL parts promotes their integration at runtime with others ubiquitous FCLs.

In our FCL (cf. Figure 1), the Controller encapsulates the functionalities required for monitoring, analyzing and planning. This means that the Controller detects the presence of new services, collects the information from the mobile devices (that join and leave the environment), processes the retrieved information and decides the required reconfigurations of the context-aware applications. These applications can be either deployed on the mobile devices or be one of the available services in the environment (*e.g.*, Multimedia Server). Consequently, the Controller requires to dynamically locate the service that operates reconfigurations of the context-aware applications. In particular, the mobile device and Multimedia Server enclose the execution part of our FCL. Moreover, the mobile device also hosts monitoring responsibilities since it notifies the Controller when changes in the provided context information occur (*e.g.*, the battery level decreases or increases). Thus, the mobility of the different elements (mobile devices and services) in the FLC makes necessary the definition of ubiquitous FCL. The next section introduces the required bindings in SCA in order to deal with this mobility.

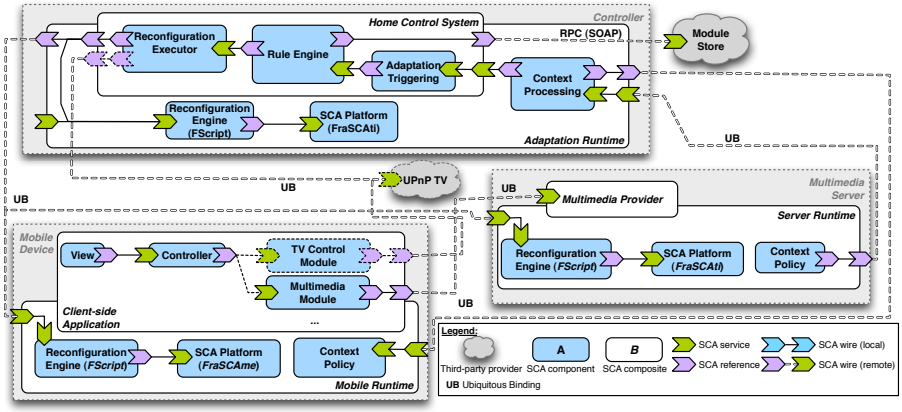

Fig. 1. Ubiquitous Feedback Control Loop for the smart home scenario.

5 Discovery of Ubiquitous Services

As already mentioned, one of the challenges for context-aware applications is the mobility. In this paper, we tackle this issue by defining a new type of binding for the SCA component model: *ubiquitous bindings*. These bindings provide a simple and lightweight mechanism for communications and promote a low coupling between the interacting entities. Furthermore, following the SCA principles, the

ubiquitous bindings enable the management of the SCA service discovery in a transparent way. These advantages make the ubiquitous bindings a suitable solution to deal with dynamicity in our ubiquitous FCL. In the rest of this section we present the design (cf. section 5.1) and implementation (cf. section 5.2) of ubiquitous bindings enabling the discovery of SCA services.

5.1 Ubiquitous Bindings

In ubiquitous environments, services constantly join and leave. For this reason, we need to provide our SCA-based FCLs with the functionality required to deal with this dynamicity. In order to introduce spontaneous interoperable communications [1,12] in SCA, we define the concept of ubiquitous binding. This new type of binding integrates state-of-the-art *Service Discovery Protocols* (SDPs) and enable the establishment of communication wires at runtime. To do that, we consider three design aspects of the SDPs [12]: *i) provider invocation*, *ii) description and attribute definition*, and *iii) provider selection*. Regarding the invocation, some SDPs (*e.g.*, UPnP [13]) define the communication mechanism. However, this mechanism is not always the most suitable. Therefore, an ubiquitous binding advertises a service provided by the SCA component as being accessible via the different SCA bindings associated with it. On the other hand, we are interested in the service description and provider selection because we need to choose the service provider according to the costumer requirements. Hence, we benefit from the discovery protocol flexibility to define properties associated with the QoS (*Quality of Service*) or QoC (*Quality of Context*) attributes (in the case of context-aware applications) [14] in the service advertisements. For defining the filters allowing provider matching, we use LDAP filters [15].

Figure 2 depicts the definition of the ubiquitous bindings for services (left side) and references (right side). The Discovery_Protocol is the name of the discovery protocol associated to the binding. The definition of an ubiquitous binding has the filter attribute in the client-side. This attribute specifies an LDAP filter that expresses restrictions of the required service in terms of its properties. In the server-side, the ubiquitous binding can have properties that provide additional information about the service, such as QoC attributes. Each property is described by the property element. By defining the ubiquitous bindings in this way, we can support the discovery of SCA context services via different discovery protocols and then access the services using the most suitable communication protocol according to the application needs. The lower part of Figure 2 shows examples of an ubiquitous binding with the SLP [16] protocol. The *precision* and *probabilityOfCorrection* are QoC attributes [17] that describe the context provided by the service. These attributes and the *contextType* properties are used in the definition of the LDAP filter in the reference. The bindings in the service side correspond to the different communication that can be used to access the context information.

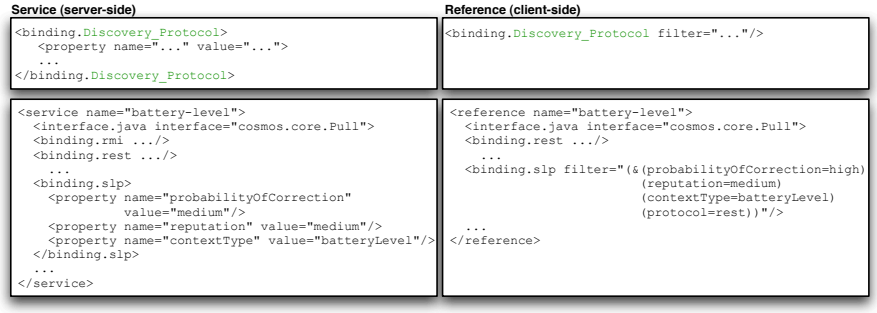

Fig. 2. SCA definition of the ubiquitous bindings.

5.2 Implementation of the Ubiquitous Bindings in the FraSCAti platform

We have integrated our ubiquitous bindings into the FraSCAti [7] platform. The FraSCAti selection is motived by two main reasons: *i)* the reflective capabilities that it introduces in the SCA programming model to allow dynamic introspection and reconfiguration of SCA based context consumers and producers, and *ii)* we can run the light version of platform (FraSCAme) on the mobile devices with limited capabilities [11].

Figure 3 depicts the integration of our ubiquitous bindings into FraSCAti. As it can be seen, an ubiquitous binding is composed of the Discoverer and Advertiser components. The Discoverer plays the role of a stub in a traditional FraSCAti binding [18]. In other words, a reference of the client component is connected to the Discoverer and is responsible for providing access to the remote SCA services. In addition to that, the Discoverer enables SCA components to search the required services at runtime. When the service is detected and selected, the *Discoverer* component provides access to it. At the server side, the Advertiser (or the skeleton in the FraSCAti terminology) publishes the services whose bindings are declared as ubiquitous. Both of them, the Discoverer and the Advertiser, are associated with a specific discovery protocol (*e.g.*, UPnP or SLP). The proposed architecture for these components modularizes different concerns of service discovery (*i.e.*, search, selection, and provider monitoring) and introduces some optional optimizations (in the Discoverer case). In this way we foster the reuse of the different components (in particular for the selection of providers), the flexibility to use different implementations and choose the required components (not all the components are mandatory). In the following sections, we present the detailed architecture of the Discoverer and Advertiser components.

Discoverer Component. This component is associated with a specific SCA reference. In order to reduce the memory footprint, we externalize the compo-

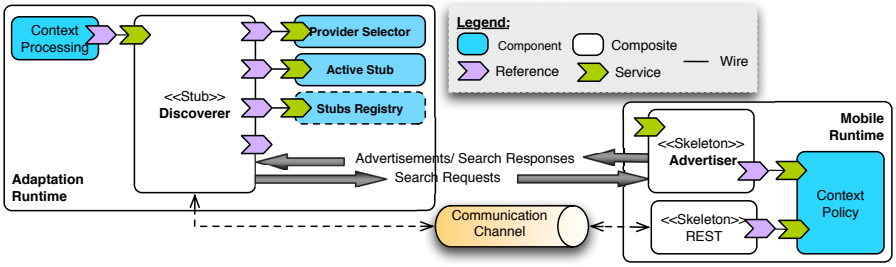

Fig. 3. Integration of ubiquitous bindings into FraSCAti.

nents providing common functionality to different discoverers. In particular, different implementations of the discoverer can share the components for provider selection, the active stubs (that encapsulate the communication with the remote service) and the stub registry (that keeps a list of the stubs already instantiated). The Discoverer Component (left side in Figure 4) has a Discoverer Orchestrator that coordinates the discovery of the requested SCA service. The Finder component sends the requests to detect the potential providers in the environment. If the filters with attributes are supported (*e.g.*, SLP), the Finder translates the LDAP filters to the protocol scheme. If the SDP does not support automatic service selection and it is needed (*e.g.*, UPnP), the Finder uses a provider selector for choosing the service. When the provider is selected, the Discovery Orchestrator disables the Finder and uses the Provider Monitor for monitoring the service availability. When the service is invoked the first time, the Discovery Orchestrator verifies in the stubs registry if there is a stub for the service provider. When this happens, the Discovery Orchestrator selects the registered stub as Active Stub. Otherwise, the Orchestrator uses the FraSCAti binding factory [11] (which is used to create wires and binding in the platform) in order to instantiate and configure the Active Stub. When the provider becomes unavailable, the Provider Monitor notifies the Orchestrator that actives again the Finder and asks it to find a new provider.

Advertiser Component. The Advertiser (right side in Figure 4) contains a component Promoter with the following responsibilities:

1. Advertise the available SCA services in the Service Registry. Each entry in the Service Registry contains the required information for the published service (*e.g.*, name, type) that is required to advertise the service. This information can be updated at runtime.
2. Listen and process search requests.
3. Notify events associated with the SCA component state.

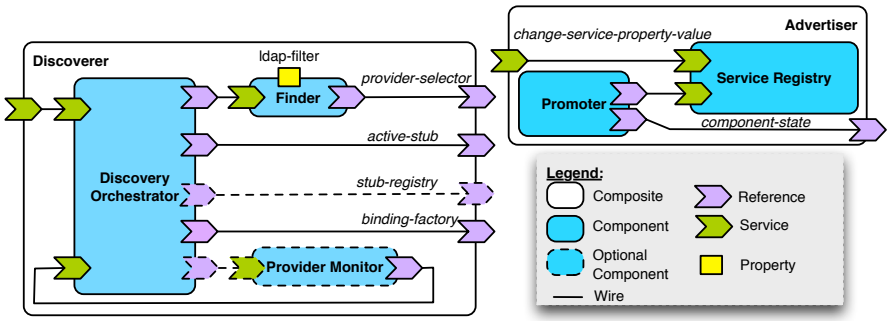

Fig. 4. Discoverer and Advertiser Architecture.

A given SCA component contains one Advertiser of an ubiquitous binding type. This means that all the services with a same ubiquitous binding type may be exported using the same advertiser instance.

Implementation Details. We have implemented ubiquitous bindings for SLP and UPnP. For the discovery via UPnP, we use Cyberlink for Java[1] version 1.7 and for SLP the jSLP library[2]. Our RESTful bindings are based in the Comanche[3] web server [19]. Both FraSCAti zand Comanche are based on the Fractal component model and use the Julia[4] implementation of the Fractal runtime environment [19].

Synthesis: The ubiquitous bindings provide a flexible and simple mechanism that allows a transparent management of mobility in our ubiquitous FCLs. These bindings leverage on the clear separation of concerns promoted by the SCA component model to avoid impact the business logic. On the other hand, the modularity of the SCA architecture of our bindings promotes their reuse and the flexibility to select the more suitable implementations of the different components.

6 Empirical Validation

To evaluate the performance of ubiquitous bindings, we implemented the scene 1 of the smart home scenario (cf. section 2.1). We tested several configurations of the scenario using two Dell Latitude 430 laptops, with the following software and hardware configurations: 1.33 GHz processor, 2 GB of RAM, Intel Pro wireless

[1] Cyberlink for Java: `http://cgupnpjava.sourceforge.net/`
[2] jSLP: `http://jslp.sourceforge.net/`
[3] Comanche web server: `http://fractal.ow2.org/tutorials/comanche.html`
[4] Julia: `http://fractal.ow2.org/julia`

3945ABG card, WIndows XP SP3, Java Virtual Machine 1.6.0_14, Julia 2.5.2 and FraSCAti 1.2. The mobile clients are two Nokia N800 Internet Table with 400 Mhz, 128 MB of RAM, interface WLAN 802.11 b/e/g, Linux Maemo (kernel 2.6.21), CACAOVM Java Virtual Machine 0.99.4, Julia 2.5.2 and FraSCAti 1.2. We also evaluate the RESTful bindings using XML, JSON and the Java Object Serialization for context representations. We used the library Xerces2 Java Parser 2.9.1 [5] for XML, and JSON-lib 2.2.34 [6] to serialize the information as JSON documents. We used SCA services to simulate the home sensors and actuators. These services are always executed on the same Dell Latitude hosting the server part of the scenario.

Providers Configuration	Provider	Discovery Latency		Retrieval Latency		
		SLP (ms)	UPnP (ms)	Object (ms)	JSON (ms)	XML (ms)
a) 1 Local Providers	N/A	68	73	244	304	315
b) 1 External Provider	Laptop	91	111	292	252	261
c) 1 External Provider	N800	216	284	513	817	818
d) 2 External Providers	Laptop & N800	507	547	576	839	845
e) 2 External Providers	N800 A & B	736	769	641	989	1046

Table 1. Performances of RESTful bindings

Table 1 summarizes the overhead observed for discovery via the ubiquitous bindings. This time includes the discovery, instantiation, and configuration of the SCA wires. The given measures are the average of 10.000 successful tests, of which the first 100 were considered as part of the warm-up. We have reduced the UPnP execution time avoiding the recovery of the service description file. We do not need this file because the advertisement messages already contain the properties required to select the provider. Regarding the discovery cost, we observe that it is possible to integrate the ubiquitous bindings in the feedback control loops with a reasonable overhead ($68ms$ per message). We also notice that the network increments the discovery latency approximately 25%, comparing the tests with a local provider (configuration a) and the laptop as provider (configuration b). Although the measures with mobile devices (configuration c, d and e) demonstrate that we can discovery services in a rational time, their use as providers considerably increase the discovery latency. This additional cost is mainly due to the limited processing capacity of these devices. As expected, SLP is more efficient than UPnP, as it is a higher-level protocol.

Table 1 also reports the costs of interactions once the services are discovered in the feedback control loop. In these tests, we use our RESTful bindings (cf. section 3.3) and three different representations for information retrieval (Java Object Serialization, JSON and XML) for the communication. As it can be seen, the exchange the information costs $244ms$ per message (configuration a). In the

[5] Xerces2 Java Parser: `http://xerces.apache.org/xerces2-j/`
[6] JSON-lib: `http://json-lib.sourceforge.net`

case of configurations including the mobile device (c, d and e), we again observe the additional overhead.

7 Related Work

In this section we present some works that deal with service discovery in ubiquitous environments. INDISS [2] (*INteroperable DIscovery System for network Services*) is a system based on event-based parsing techniques to provide full service discovery interoperability. The authors claim that this interoperability is achieved without altering existing applications and services. INDISS exploits the multicast groups used by different discovered protocols to detect the protocols being used in the environment. Then, INDISS transforms the SDP messages to events that will be transformed again into messages that correspond to the SDP supported by the client application. Although interoperability between discovery protocols is an interesting solution for the mobility problem, the applications have yet to use always the same communication protocols defined by the SDP even if it is not suitable. Our SCA-based solution provides the required flexibility for client (resp. server) applications can search (resp. advertiser) the required (resp. provided) services using the more suitable discovery and interaction protocols. In this way, the devices only deploy the required functionality.

In [20], authors propose a framework for the development of an adaptive multi-personality service discovery middleware, which operate in fixed and adhoc networks. According to authors, the framework promotes component re-use and simplifies configuration and dynamic reconfiguration of multiple concurrent protocols. With our ubiquitous bindings we also foster re-use and dynamic reconfiguration capabilities thanks to the combination of the SCA component model and the FraSCAti platform.

ReMMoC (Reflective Middleware for Mobile Computing) [4] is an adaptive middleware for discovery and access of services in mobile clients. According to authors, ReMMoC reconfigures itself to use the current discovery protocols present in the environment. Furthermore the middleware interoperate with services implemented upon different interaction types. Although in our approach we do not deal with the adaptation issue in the communication level, by using the reconfigurability capabilities offered by FraSCAti and our ubiquitous feedback control loops, we could identify situations where new discovery or interaction bindings are required and deploy them on the clients. Furthermore, the integration of ubiquitous bindings in SCA promotes the use of these bindings in any SCA application not only mobile clients.

8 Conclusions and Perspectives

In order to deal with the mobility issue in ubiquitous environments, we have enabled spontaneous communications in the SCA-based applications. To do it, in this paper, we define ubiquitous bindings, a new kind of binding for the SCA

standard that allows the integration at runtime of service providers and consumers. Our ubiquitous bindings advertise and discover services via different discovery protocols and select them applying LDAP filters. The flexibility of the ubiquitous bindings allows the service access using the SCA traditional bindings associated with the services. Furthermore, the design of the ubiquitous bindings is based on SCA to enable their integration in any SCA runtime platform. By benefiting from SCA extensibility and its clear separation of concerns, we integrate in applications discovery management in a transparent way. Thus, the originality of our solution rests on its simplicity and efficiency achieved by the combination of well defined and accepted standards and protocols.

To illustrate the use of ubiquitous bindings, we define ubiquitous FCLs enabling adaptation of context-aware applications. The exchange of context information in these FCLs is achieved via RESTful bindings that allow us to face heterogeneity in ubiquitous environments. The ubiquitous bindings have been integrated into the FraSCAti platform, following an architecture that promotes the sharing of common functionality of service discovery. The suitability of our ubiquitous bindings was confirmed with tests executed using a smart home scenario.

Future work includes further tests using different kinds of mobile devices, protocols and service providers. In the particular case of ubiquitous FCLs, we plan to improve the performance of our solution by introducing a cache mechanism that enables the temporal storing of the retrieved context information. In this way, when all the required information is gathered, it can be processed even if any of the context providers is not available (or the connection was lost). Finally, we will exploit the introspection and reconfiguration capabilities brought into SCA by the FraSCAti platform in order to instrument the adaptation process in the mobile devices via our RESTful bindings.

References

1. Kindberg, T., Fox, A.: System software for ubiquitous computing. IEEE Pervasive Computing **1**(1) (2002) 70–81
2. Bromberg, Y.D., Issarny, V.: Indiss: interoperable discovery system for networked services. In: Middleware '05: Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware, New York, NY, USA, Springer-Verlag New York, Inc. (2005) 164–183
3. Nakazawa, J., Tokuda, H., Edwards, W.K., Ramachandran, U.: A bridging framework for universal interoperability in pervasive systems. In: ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, Washington, DC, USA, IEEE Computer Society (2006) 3
4. Grace, P., Blair, G.S., Samuel, S.: A reflective framework for discovery and interaction in heterogeneous mobile environments. SIGMOBILE Mob. Comput. Commun. Rev. **9**(1) (2005) 2–14
5. Beisiegel, M. et al: Service Component Architecture (November 2007)
6. Hariri, S., Khargharia, B., Chen, H., Yang, J., Zhang, Y., Parashar, M., Liu, H.: The Autonomic Computing Paradigm. Cluster Computing **9**(1) (2006) 5–17

7. Seinturier, L., Merle, P., Fournier, D., Dolet, N., Schiavoni, V., Stefani, J.B.: Reconfigurable sca applications with the frascati platform. In: SCC '09: Proceedings of the 2009 IEEE International Conference on Services Computing, Washington, DC, USA, IEEE Computer Society (2009) 268–275
8. Parashar, M., Hariri, S.: Autonomic Computing: An Overview. Unconventional Programming Paradigms (2005) 257–269
9. Open SOA: SCA Transaction Policy. (December 2007) Version 1.0.
10. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
11. Romero, D., Rouvoy, R., Seinturier, L., Chabridon, S., Denis, C., Nicolas, P.: Enabling Context-Aware Web Services: A Middleware Approach for Ubiquitous Environments. In Michael Sheng, Jian Yu, Schahram Dustdar, eds.: Enabling Context-Aware Web Services: Methods, Architectures, and Technologies. Chapman and Hall/CRC (07 2009)
12. Zhu, F., Mutka, M.W., Ni, L.M.: Service discovery in pervasive computing environments. IEEE Pervasive Computing **4**(4) (2005) 81–90
13. UPnP Forum: UPnP Device Architecture 1.0. `http://www.upnp.org/resources/documents.asp` (april 2008)
14. Krause, M., Hochstatter, I.: Challenges in Modelling and Using Quality of Context (QoC). In: Proceedings of the 2nd International Workshop on Mobility Aware Technologies and Applications, Montreal, Canada (2005) 324–333
15. Smith, M., Howes, T.: RFC 4515 - Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters. IETF RFC (1996)
16. Guttman, E., Perkins, C., Veizades, J., Day, M.: Service Location Protocol, Version 2. RFC 2608 (Proposed Standard). `http://tools.ietf.org/html/rfc2608` (june 1999)
17. Dey, A.K., Abowd, G.D., Salber, D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Hum.-Comput. Interact. **16**(2) (2001) 97–166
18. SCOrWare Project: SCA Platform Specifications - Version 1.0 (2007)
19. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The Fractal component model and its support in Java. Software: Practice and Experience – Special issue on Experiences with Auto-adaptive and Reconfigurable Systems **36**(11-12) (August 2006) 1257–1284 John Wiley & Sons.
20. Flores-Cortés, C.A., Blair, G.S., Grace, P.: A multi-protocol framework for ad-hoc service discovery. In: MPAC '06: Proceedings of the 4th international workshop on Middleware for Pervasive and Ad-Hoc Computing, New York, NY, USA, ACM (2006) 10