Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 7 (2012) pp. 316-323
© MIR Labs, www.mirlabs.net/jias/index.html

A Semantic-Enhanced Distributed Architecture for Providing and Consuming Web Services using DPWS

Mehdi Khouja[1], **Carlos Juiz**[1], **Ramon Puigjaner**[1] **and Farouk Kamoun**[2]

[1]Department of Mathematics and Computer Science, University of the Balearic Islands
Cra. de Valldemossa, km 7.5. 07122, Palma de Mallorca, Spain
{*mehdi.khouja,cjuiz,putxi*}*@uib.es*

[2]National School of Computer Science, University of La Manouba
La Manouba University Campus, 2010, La Manouba, Tunisia
frk.kamoun@planet.tn

Abstract: **The omnipresence of mobile devices has changed the users behaviour. The content present in the Web is created by simple users. Nevertheless, this content is in the cloud and not shared directly from users. This behaviour is a passive one. In this paper, we propose a distributed service architecture which transforms passive users into service providers. This role is acquired in a specific ambient where users share services with each other. Thanks to this collaboration, people can find adequate solutions to a specific problem. A university campus scenario illustrates the functionalities of the service architecture. The proposed solution has a layered structure that integrates Web services with a semantic modelling of the context. This model describes a specific vocabulary to be used when creating or requesting services. The resulting services are organised in groups according to selected criteria. The solution is based on the Device Profile for Web service (DPWS).**

Keywords: context-awarenes, web service, user behaviour, service discovery, dpws

I. Introduction

Users have a passive behaviour when interacting with their mobile phones. They only consume services provided by third parties. An active behaviour is possible. It will transform users into service providers. In fact, people localised within the same ambient may share common interests and collaborate to solve problems.
Let's consider the following scenario. John, a student in biology, is searching for a solution to a homework. He starts a mobile application to search for help within the university community. Via a guided wizard, John specifies the characteristics of the problem he is facing. He selects the filed, the category and the type of the problem. He can also indicates the type of solution he is searching. It can be a file, a link to a Web site, a book reference or a meeting date with whom has found the solution. Once John has finished specifying his needs, the mobile application transforms them to a web service request.
On the other hand, Bill has found a solution to the above problem. He starts the same application as John, but in this case with a provider profile. Via a guided wizard, he specifies the solution he is offering. John's mobile has only to discover the announced service and access the solution. Other students may behave like John or Bill. In this case the campus network will be overcharged with service requests and announces. Also, the users may be confused when searching for the adequate solution. Organising services into groups is the solution to these issues. In this case the provider profile has to integrate a more complex behaviour. Once Bill has specified his solution, the application tries to find similar ones. This list can be accessed as a web service as well. Once such a list is discovered, Bill service is aggregated to that list. Otherwise, a new provider list is created. From this point, we have to distinguish between two provider profiles: the simple one and the leader profile. The first one waits for service request and performs the service invocation when required. The second profile maintains the provider list by adding and removing elements. When a leader leaves the ambient, he has to assign a new leader. The later has to notify the providers about the leader change.
Realising such a scenario is done through a service oriented architecture. To accomplish this aim, various steps need to be taken. First, a stack of protocols has to be selected in order to implement a service oriented architecture. Then, a context model has to be designed. This model describes the shared characteristics among the context elements. The behaviour of the elements operating within the ambient is the next step in the design of the proposed architecture. It establishes the users organisation as well as their interactions. It specifies the way the services are announced and requested within the ambient.
This paper is organised as follow. First, works dealing with service oriented architecture and context awareness are described. The proposed solution is presented in section III. The design process is detailed. Then, the architecture components are defined. Section IV deals with user behaviours. It shows the activities associated with the profiles of requester

and provider. Finally, the Web service generation process is illustrated.

II. Related Work

Various context-aware systems have been developed. Some of them adopt the service oriented paradigm. The discussion is focused on the user behaviour: passive or active.
CoWSAMI is a service-oriented middleware platform [1]. It supports context-awareness in pervasive environments. It provides a context manager which is responsible for maintaining heterogeneous context sources. The context informations are collected through services. The users are passive since they only provide the platform with context information to have personalised services.
The Anyserver platform [2] is a client-proxy-server architecture that supports context-awareness. The context takes into account device, network and application characteristics. The mobile users are only consumers. They access application provided by the AnyServer platform.
In [3], a context framework is presented. It aims to facilitate the development of context-aware adaptable Web service. The context information is related mainly to user characteristics such as location and profile. The user behaviour is a passive one. In fact, it only consumes Web service.
SOCAM [4] is middleware for building context-aware services. The context model presents a generalised ontology and a specific domain one. The later describes various context such as home or vehicle. The service discovery is registry based. Users access personalised services according to their context. They are not service providers, but only context providers.
CA-SOA [5] is a context-aware service oriented architecture. It describes context via an ontology-based model. The architecture components support an ubiquitous discovery and access of Web services. The service repository has a centralized structure. Therefore, it can not support a distributed environment.
In [6] Truong et al. survey web service-based context-aware systems.

III. Distributed Service Architecture

In this section the design steps for the proposed architecture. First, the service discovery process is detailed. Then, the context model is presented. Finally, the architecture composition is described.

A. Service Discovery Model

The first step is designing the service oriented architecture. Hence, three core concepts have to be specified: Requesting, providing and discovering 1. Among these concepts the later is a key one. It determines the communication model within the architecture components: centralised or distributed. In the first case, a central repository is dedicated to store the different available services. Whether in the second one, each provider is responsible for announcing his services. Since we are dealing with services in a mobile environment, a distributed architecture has to be adopted. Another issue in designing a SOA is related to specification choices: standard or proprietary.

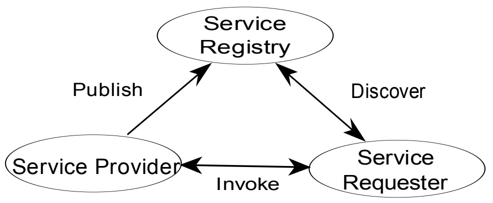

Figure. 1: Service oriented architecture core elements

Various service discovery model standards have to be analysed in order to find the appropriate one for our proposed approach.
Jini [11], originally developed by Sun, offers a service oriented framework for constructing distributed systems. The goal of Jini architecture is the federation of groups of clients/services within a dynamic computing system. Jini enables users to share services and resources over a network. The technology infrastructure is Java technology centred. The discovery process relies on a central registry called a look up service (LUS). Due to its Java dependency and centralised approach, Jini does not meet our requirements.
The Service Location Protocol [12] (SLP) is being developed by the IETF. It provides a scalable framework for the discovery and selection of network services.
UPnP [13] (Universal Plug and Play) is maintained by the UPnP forum initiative. It aims to offer a seamless connectivity to devices within a network. It comes with a set of specification defining the addressing and the discovery of resources as well as the description and the control of the services within the network. The UPnP discovery process is base on the Simple Service Discovery Protocol (SSDP). The process is directory-less.
Bluetooth comes with its own protocol stack. As part of it, it offers its proper service discovery methods: Bluetooth Service Discover Protocol (Bluetooth SDP) [14]. The SDP specify the behaviour a Bluetooth client application in order to discover the available services in the Bluetooth servers.
The device profile for web service [7] (DPWS) is an approved standard of the organization for the advancement of structured information standards (OASIS) . It is a stack of WS-standards that enables Web service capabilities on resource-constrained devices. DPWS allows sending secure messages to and from Web services, discovering a Web service dynamically, describing a Web service, subscribing to, and receiving events from a Web service. It defines the following components:

- Client: A network endpoint that sends and/or receives messages from a service.
- Service: A software system that exposes its capabilities by receiving and/or sending messages on one or several network endpoints.
- Device: A distinguished type of service that hosts other services and sends and/or receives one or more specific types of messages.

Taking into account the architecture requirements of the described case of study: distributed environment and resource-constrained devices , DPWS is adequate for the proposed solution.
Figure 2 illustrates the DPWS protocol stack.

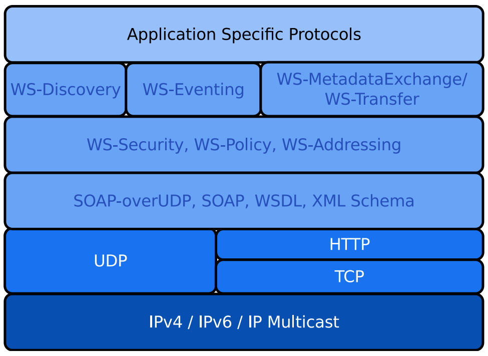
Application Specific Protocols
WS-Discovery
WS-Eventing
WS-MetadataExchange/ WS-Transfer
WS-Security, WS-Policy, WS-Addressing
SOAP-overUDP, SOAP, WSDL, XML Schema
UDP
HTTP
TCP
IPv4 / IPv6 / IP Multicast

Figure. 2: The device profile for web services protocol stack [8]

- WS-Addressing [15]: It describes addressing information in SOAP message header independently from the transport protocol.
- WS-Security [21]: It allows secure communication to Web services by providing mechanisms of signing, encryption and identity authentication.
- WS-Policy [19]: It allows services to express their policies (security, quality of service) and clients to specify their needs.
- WS-Metada-Exchange [18]: I specifies meta-data description of hosted services and hosting devices.
- WS-Eventing [17]: It defines a protocol of event notifications via a subscription mechanism.
- WS-Discovery [16]: It defines a multicast discovery protocol to locate services. It supports various matching techniques between the request and the service. These techniques include URI (Uniform Resource Identifier), UUID (Universally-unique Identifier), LDAP (Lightweight Directory Access Protocol) and case-sensitive comparison.

WS-discovery defines two operational mode: an ad-hoc mode and a managed mode. The first one consists on a distributed communication model. Clients and devices communicates via multicast and unicast messages. The second one involves a device proxy that facilitated the discovery of services. The ad-hoc mode is appropriate for our distributed architecture. The discovery process is composed of various message exchanges. When a device joins the network, it sends a multicast Hello (1) announcing itself and its hosted services. A Client listens for multicast Hello. If a client misses the later message, it sends a multicast Probe (2) to locate a specific device and/or service. If a device matches the Probe message, it sends a unicast Probe Match message (3). If the transport address of the device was not included in previous transactions, the client sends a multicast Resolve message (4) including the device identifier. The corresponding device answers with a unicast Resolve Match message (5) with its transport address. At this point the client has discovered the adequate service for its request. The client initiates the meta-data exchange phase in order to get the corresponding meta-data of the device (6,7) and the searched service (8,9). This is fulfilled via the WS-Meta-data-Exchange and WS-Transfer [20] protocols. Once the client has the meta-data, it can invoke the desired service (10,11). When a device leaves the network, it sends a multicast Bye message (12). Figure 3 shows the message exchanges during the service discovery phase in DPWS.
The analysis of the discovery process shows that a client has to know the exact service he is searching. Otherwise he has to treat all the discovered services. Hence, the users need to share a vocabulary in order to describe the service semantic. Therefore, users will employ the same words when generating services. The vocabulary is related to the context where the clients are located. In our case, it describes a problem/solution issue within the university community. In the next section, the model of the context vocabulary is detailed.

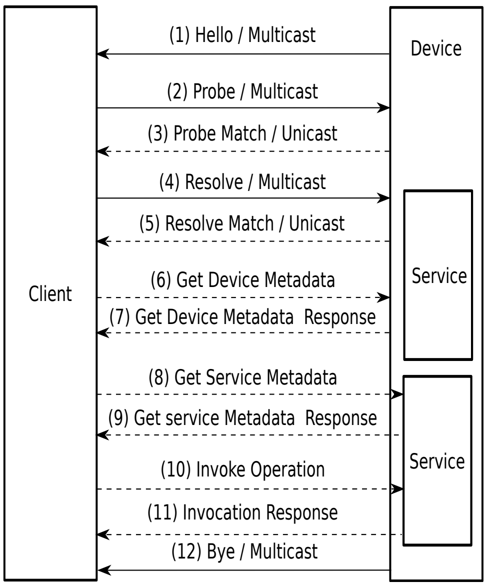

Figure. 3: Message exchanges in DPWS

B. *Service Context Model*

The second step is modelling the context in which operate the users. The context information cover a wide range of characteristics. Dey's [10] context definition summarises these different information sources: *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

Since we are dealing with service architecture, the information related to the provided service will be taken into account. In our case, the information include users' profiles, the prob-

lems and the solutions. Not only these elements have to be represented, but also the relationships between them.
Various modelling techniques exist in the literature: key-value, markup scheme, graphical, object oriented, logic-based models and ontology-based models. The last one fits our modelling requirements. In fact, ontologies represent a description of the concepts, properties and relationships between concepts. They have a high and formal expressiveness. Knowledge sharing and reuse can be performed between heterogeneous context sources. A discussion of the different modelling techniques is presented in [9].
The user behaviour has to reflect the SOA concepts. A user can be a requester or a provider. The first one task is to generate the service request corresponding to the user choices and searches for the adequate response. The provider is responsible for organising the different services available within the ambient. This task consists on arranging the service among group following specific criteria that describe the services.

C. Layered Service Oriented Architecture

In a specific context, each user is identified as a Context Element (CoxEl). The CoxEl is structured in a layered architecture. Three principles layers compose the CoxEl: Web service, semantic and application layer. Figure 4 shows the layered architecture.

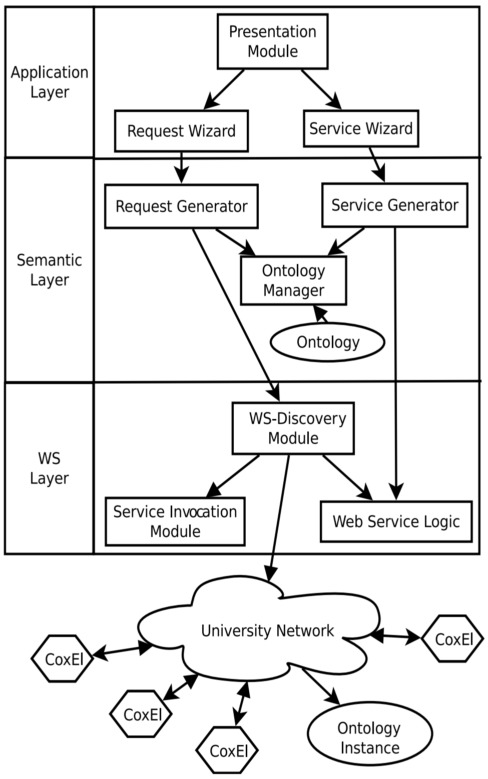

Figure. 4: CoxEl layered architecture

- The application layer consists on the application presented in the motivation scenario. It includes three components: the presentation module, the request wizard and the service wizard. The first one is the user interface. The wizard modules retrieve the vocabulary necessary for generating the request or the service.
- The semantic layer includes the problem statement ontology. This ontology describes the vocabulary to be used when requesting or providing the Web services. This layer is an intermediate between the application and Web service layer. Once the different concepts related to the problem or solution description are chosen, they are transmitted to the Web service layer. Two generator module are responsible of this task. An ontology manager module is part of this layer. Its task consists on importing the ontology instance into the semantic layer. It updates the ontology according the profile chosen for the CoxEL. Figure 5 illustrates the problem statement ontology. In the motivation scenario some concepts will be instantiated in order to reflect the campus domain. A user can be a student, a professor or a staff member. Its role depends on the profile he uses the application. He can express a solution or a problem. These two petitions are described using the field, category and type concepts. In the campus environment, the field corresponds to the different discipline in the university: biology, physics, chemistry, computing,... The category concept is instantiated as practical or theoretical. The type can be an exercise, a specific question or a chapter reference. The solution is provided through a resource such as an Web site, a file, a text message or a meeting date.
- The Web service layer incorporates SOA core functionalities: discovery, invoking and service description. The web service and its corresponding operations are created according to the choices made by the user and the transformation operated by the service generator in the upper layers. The WS-layer incorporates the DPWS stack in order to implement a WS oriented architecture.

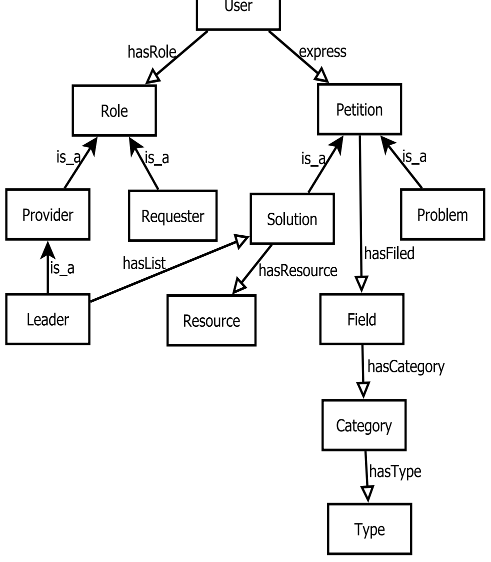

Figure. 5: Problem statement ontology

IV. Context Elements Behaviours

The context elements behaviour depends on the profile used with the application. A CoxEl can be a provider or a requester. In this section, the activities corresponding to each behaviour are detailed.

A. Requester Behaviour

The requester behaviour has a simple sequence of activities. Figure 6 shows the activity diagram corresponding to a CoxEl requester. First, the problem statement ontology instance is loaded within the semantic layer. The vocabulary described by the instance and the user choices via the application are inputs for generating the service request. Since the service are organised in groups, the requester searches for the provider list service. If such service is discovered, the CoxEl will choose the adequate solution service to invoke.

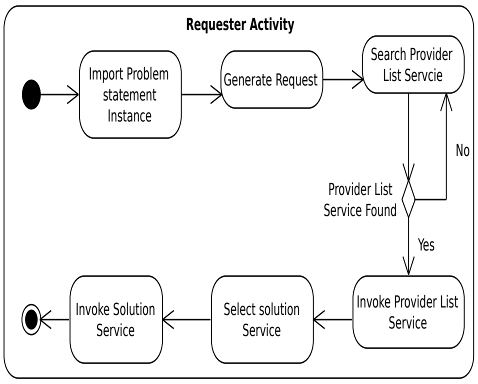
Requester Activity
Import Problem statement Instance
Generate Request
Search Provider List Servcie
No
Provider List Service Found
Yes
Invoke Provider List Service
Select solution Service
Invoke Solution Service

Figure. 6: Requester activity diagram

B. Provider Behaviour

The provider is responsible of announcing services and performing management tasks. The problem statement ontology instance is loaded first. Once the service is generated, the CoxEl searches for a provider list service that meets the one his is offering. Whether or not this service is found, two different scenarios can occur. The first one when the provider list service is found. Then, the CoxEl acts as a simple provider. The second scenario occurs when no provider list service is found. In this case, the CoxEl will acquire the role of leader. Figure 7 gives an overview of a provider CoxEl.
A simple provider offers one type of service related to the solution. After the start up phase, the CoxEl invokes the join service from the provider leader. Once added to the service list, it enters in a listen state. It can receive three type of requests. A solution service invocation . In this case the CoxEl responds with the solution he is offering. The two other requests concern the leadership of the provider list. The provider can be notified about a leader change. A leader assignment request converts the provider into a leader. In this case, it must notify all the providers in the list about this change. When a provider decides to leave the ambient, it

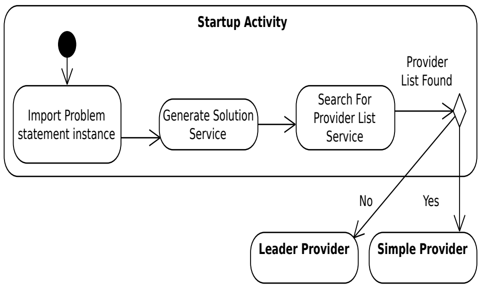
Startup Activity
Import Problem statement instance
Generate Solution Service
Search For Provider List Service
Provider List Found
No
Yes
Leader Provider
Simple Provider

Figure. 7: Provider activity diagram

must notify the leader. Figure 8 details the activities related to a simple provider

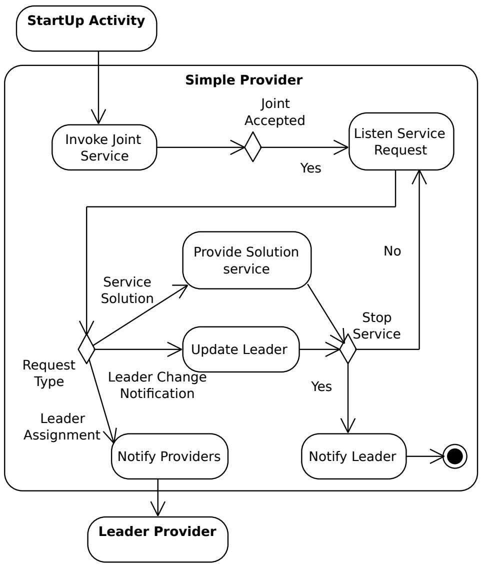

Figure. 8: Simple provider activity diagram

Figure 9 details the activities related to a leader provider. This role is responsible of maintaining a list of service provider. It starts by creating a new list and announcing it as a service within the ambient. The CoxEl enters in a listen state. It can receive three type of requests. A solution service request. In this case the CoxEl responds with the solution he is offering. A CoxEl can request joining the list. Then, the leader update the list by adding the new provider. A requester can invoke the provider list service. When a leader leaves the ambient, he must assign a new leader from the provider list.

V. Web Service Generation Process

The CoxEl provider integrates two type of services: Solution service and management service. The solution service is composed of various operation depending on the type of the solution. Let's consider our campus scenario. The generation process from the vocabulary selection to Web service is described in figure 10. The figure shows the ontology concepts with their corresponding individuals related to the campus domain. The marked individual-

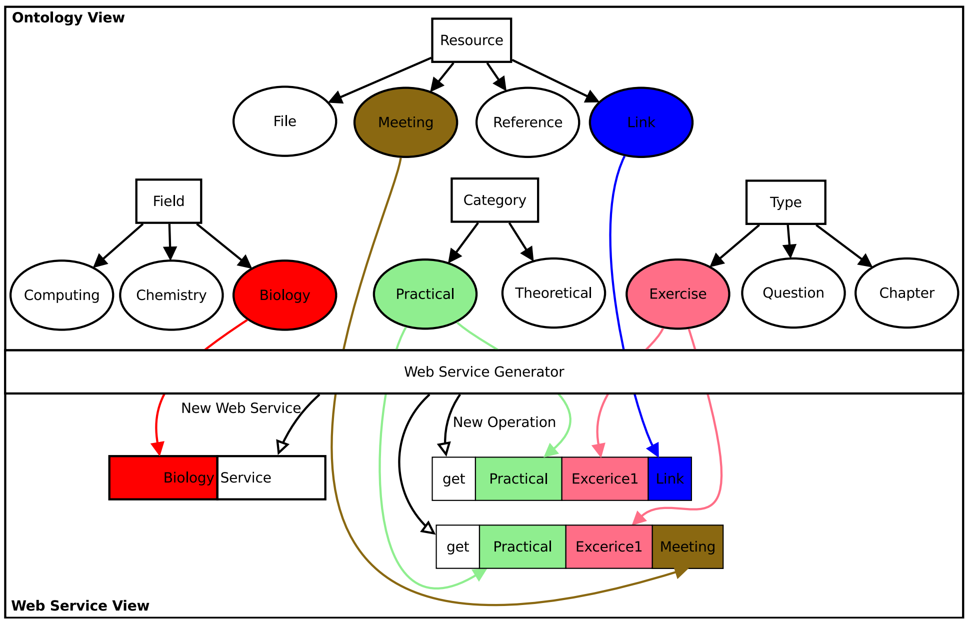

Figure. 10: Web service generator for a simple provider

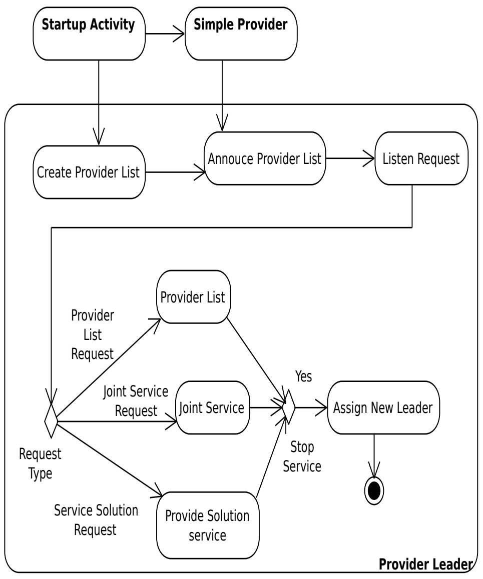

Figure. 9: Leader provider activity diagram

s are the ones chosen by Bill when specifying his solution. The choices are biology for field, practical for category, exercise for type and meeting and link for resources. A Web service *BiologySolution* is created. Two operations are associated to it: *getPracticalExerciseOneMeeting* and *getPracticalExerciseOneLink*. The first one return a date and a string that corresponds to the location. The output of the second operation is a Web site link.
In case the CoxEl has a leader profile, it will integrate management services. These services include a join service and a list of provider service. In our motivation scenario, if Bill's device has a leader role, it will create a new Web service. The provider list may corresponds to a list of service or specific operation. For example a service called *ProviderListBiology* will return all the solutions corresponding to the biology field. The service *ProviderListBiologyPractical* will return all the service operations that match the biology and practical criteria. The leader implements a *JoinService* that is responsible of maintaining the list of providers. It includes operations such as *addProvider*, *removeProvider* and *updateProvider*.

VI. Conclusion and Future Work

In this paper, we presented an approach that transforms passive users into service provider. The solution is designed as a service oriented architecture with a layered structure. The main characteristic is the use of an ontology model to describe the vocabulary used in service specification and requesting. Also, it integrates Web services standards. The services are organised within groups according to specific criteria. This solution limits the message exchanges among the users. We illustrated the process of service generation for the provider profile. Extrapolating the architecture to another domain is done by creating an instance of the problem/statement ontology. Considering a hospital case of study, the ontology vocabulary will describe service related to this specific scenario. The architecture can switch from a scenario to another depending on the ontology instance available for the user context. The application prototype is an ongoing work. Once, it is finalised, we will move to the test phase. First, an emulation is necessary in order to evaluate the grouping feature. Then in a real world test, a user satisfaction metric will be measured.

Acknowledgement

This work is partially supported by the project *TIN2007-60440 Arquitectura Semántica Orientada a Servicios (SOSA)* under *Programa Nacional de Tecnologías Informáticas* of Ministerio de Ciencia e Innovación, Spain. It has been selected under an operational program financed by the European Social Fund. Mehdi Khouja is a scholarship holder of the Government of the Balearic Islands, Ministry of Education, Culture and Universities.

References

[1] D. Athanasopoulos, V. Issarny, E. Pitoura, P. Vassiliadis, and A. Zarras, Mobile Web Services for Context-Aware Pervasive Environments, in *ACM Transactions on Internet Technology*, vol. V, no. December, pp. 1–30, 2005.

[2] B. Han, W. Jia, J. Shen, and M. Yuen, Context-awareness in mobile web services, in *Parallel and Distributed Processing and Applications*, pp.519–528, 2005.

[3] M. Keidl and A. Kemper, A framework for context-aware adaptable web services, in *Advances in Database Technology-EDBT 2004*, pp. 635–636, 2004.

[4] T. Gu, H. Pung, and D. Zhang, A service-oriented middleware for building context-aware services, in *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, 2005.

[5] I. Chen, S. Yang, and J. Zhang, Ubiquitous provision of context aware web services, in *Services Computing, 2006. SCC'06. IEEE International Conference on.* IEEE, 2006, pp.60–68.

[6] H.-L. Truong and S. Dustdar, A survey on context-aware web service systems, *International Journal of Web Information Systems*, vol. 5, no. 1, pp.5–31, Mar. 2009.

[7] OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) TC,Web services dynamic discovery (ws-discovery) version 1.1, http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html, July 2009.

[8] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski, Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services, in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07).*IEEE, 2007, pp.956–963.

[9] M. Khouja, C. Juiz, I. Lera, R. Puigjaner, and F. Kamoun, An Ontology-Based Model for a Context-Aware Service Oriented Architecture, in *Software Engineering Research and Practice*, H. R. Arabnia and H. Reza, Eds. CSREA Press, 2007, pp.608–612.

[10] Anind K. Dey. Understanding and Using Context, in *Personal Ubiquitous Computing*, 5(1), pp:4–7, 2001.

[11] Jini technology, 2009, http://www.jini.org.

[12] J. Veizades E. Guttman, C. Perkins and M. Day. Service location protocol, version 2, rfc 2608. Technical report, 1999.

[13] UPnP Forum. Universal plug and play device architecture version 1.1. Technical report, 2008.

[14] Bluetooth SIG. Service discovery application profile version 1.1. Technical report, 2001.

[15] W3C Recommendation, Web Services Addressing 1.0 - Core, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509, 9 May 2006.

[16] OASIS Standard, "Web Services Dynamic Discovery (WS-Discovery), http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.docx, 1 July 2009.

[17] D. Box, et al, Web Services Eventing (WS-Eventing), http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315/, 15 March 2006.

[18] K. Ballinger, et al, "Web Services Metadata Exchange 1.1 (WS-MetadataExchange), http://www.w3.org/Submission/2008/SUBM-WS-MetadataExchange-20080813/, 13 August 2008.

[19] W3C Recommendation, Web Services Policy 1.5 - Framework, http://www.w3.org/TR/2007/REC-ws-policy-20070904/, 4 September 2007.

[20] J. Alexander, et al, Web Service Transfer (WS-Transfer), http://www.w3.org/Submission/2006/SUBM-WS-Transfer-20060927/, 27 September 2006.

[21] OASIS Standard Specification, Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 1 February 2006.

Author Biographies

Mehdi Khouja received the engineering and M.Sc degrees in Computer Science in 2003 and 2004, respectively from The National School of Computer Science (ENSI), Tunisia. Currently, he is a PhD candidate in Computer Science at the University of Balearic Islands (UIB), Spain.

Carlos Juiz received the B.Sc., M.Sc and Ph.D. degrees in Computer Science in 1988, 1992 and 2001, respectively from the University of Balearic Islands (UIB), Spain. He is co-author of more than 140 international papers and one university textbook. Currently he is Vice rector for Information Technologies at UIB and Associate Professor at the Computer Science Department. He is Senior Member of ACM since 2010. He is also Academic Advocate of ISACA.

Ramon Puigjaner is industrial engineer (Universitat Politècnica de Catalunya (UPC), Barcelona 1964), Master in Aeronautical Sciences (Ecole Nationale Supérieure de l'Aronautique, Paris 1966). Ph.D. (UPC, 1972), License (Bachelor) in Informatics (Universidad Politècnica de Madrid, 1972) and Doctor Honoris Causa (Universidad Nacional de Asunción, Paraguay, 2010). Currently he is Emeritus Professor in Universitat de les Illes Balears. Spain. He is author of a book and of more than 200 reviewed papers in international journals and conferences. He is Life Senior Member of IEEE and Distinguished Educator of ACM.

Farouk Kamoun received a engineering degree in 1970 from l'Ecole Supérieure d'Electricité (Supélec), a Master's and a PhD degree in 1972 and 1976, respectively from the University of California at Los Angeles Computer Science Department. He is co-author of over a hundred publications in journals and scientific conferences, in the field of computer networking, internet protocols and software engineering. Currently, he is president of l'Ecole Supérieure des Sciences Appliquées et de Management (SESAME), Tunisia.