Enhanced service discovery via shared context in a distributed architecture

A dissertation presented as partial fulfilment of the requirements for the degree of Ph.D. in the subject of Computer Science

Presented by:
Mehdi Khouja

Supervised by:
Dr. Carlos Juiz García

Palma, Spain. 2013
In the name of Allah, the Most Gracious, the Most Merciful.
Dedicated to

my mother and father, Noura & Mohamed

my sister and brother, Maryem & Ahmed

my loving wife Mariem
Una tesis doctoral no es sólo esta memoria y su defensa. Una tesis son muchos años de trabajo, de buena convivencia, de encuentros, de intercambios culturales, de discusiones, de momentos compartidos y, sobre todo, de personas. Una tesis es una aventura, un capítulo de la vida. Muchos son los que han participado en esta aventura, por eso me gustaría dedicarles las siguientes palabras.

En primer lugar, me gustaría dar las gracias al Dr. Carlos Juiz por haber sido el capitán de esta aventura. Gracias a él, he adquirido mejores conocimientos y experiencias como investigador pero, especialmente, como persona. Siempre ha mostrado una total disponibilidad y no ha dudado en ofrecerme su ayuda, apoyo y compartir conmigo sus reflexiones y experiencias. Estoy muy orgulloso de trabajar con él, de haberle tenido como director de tesis pero, sobre todo, de tenerlo como amigo.

Me gustaría también agradecer a dos personas sin las que no habría podido empezar esta aventura. Agradezco al Dr. Ramon Puigjaner por haberme permitido venir a Palma para hacer mi tesis y por haberme recibido en su grupo de investigación. Je voudrais aussi remercier le Pr. Farouk Kmaoun pour m’avoir dirigé et conseillé tout au long de mon parcours universitaire.

Quiero dar las gracias a dos amigos y colegas, Issac y Carlos, por el apoyo y la ayuda que me han ofrecido y por los momentos que hemos compartido tanto a nivel personal como profesional. Quiero mostrar mi agradecimiento a los amigos que han pasado o que siguen estando en el grupo de investigación ACSIC: Pere Pau, Katya, Jaume, Beatriz, ...

Quiero también dar las gracias a mis amigos y compañeros de piso, especialmente a Emilio e Iván. Gracias Emilio por haberme aguantado todos estos años, gracias por tu apoyo y tus consejos. Gracias Iván por tu amistad y por ser un buen compañero.

No quiero olvidar a mis amigos del grupo Averroes de la UIB: Mounir, Asma, Hanan, Lamiæ, Manal, Amani y Marwen. Gracias por los momentos que hemos compartido. Gracias también a mis paisanos en Palma por sus amistades y por el ambiente agradable y fraternal que han recreado en la isla.
Esta tesis no sería posible sin las entidades que la han financiado: El Govern de les Illes Balears a través de la Conselleria d’Educació, Cultura i Universitats i la Direcció General d’Universitats, Recerca i Transferència del Coneixement (DGURT), la Agencia Española de Cooperación Internacional y Desarrollo (AECID) y la empresa Telefónica a través de la Cátedra Telefónica-UIB de Sanidad Digital y Turismo Sostenible. Gracias a los servicios administrativos de la UIB particularmente del Departament de Ciències Matemàtiques i Informàtica.

Todas estas personas tienen un lugar particular en esta aventura que es parte de una tesis más importante, la tesis de mi vida, que presentaré cuando sea el momento delante del tribunal del Justo y ojalá la apruebe.
Abstract

The objective of this thesis is to demonstrate that sharing the vocabulary for service description enhances the service discovery mechanism. The proposed solution is a distributed architecture for enhanced context-aware web services. The starting point is a motivation scenario in which university students are trying to share a solution about a specific problem in a campus environment. Taking into account the case’s requirements, two research scopes have been set: context-aware systems and service discovery mechanisms. In the first one, context modeling techniques were analyzed. System capabilities were illustrated via an abstract layered architecture. In the second scope, industry and consortia supported discovery approaches were described. A special focus was given to research initiatives that integrate the context-awareness feature within discovery mechanisms.

The proposed solution includes an ontology-based context model for describing service vocabulary. This model is shared among users to facilitate the description of their petitions. The Devices Profile for Web Services (DPWS) was integrated in the architecture as a framework for sending, describing and discovering Web services. The adopted validation methodology consisted in comparing scenarios with the context ontology as vocabulary source and others that use synonyms from Wordnet. A series of discrete-event simulations were set up by specifying performance metrics related to the discovery mechanism, control parameters and user behavior models. The results have shown that using the context ontology enhances the discovery ratio as well as the mean discovered services per request. Scenarios with the ontology as vocabulary source generated an overhead of probe messages compared to Wordnet-based scenarios.
Resumen

El objetivo de esta tesis es demostrar que los mecanismos de descubrimiento de servicio mejoran cuando se comparte el vocabulario para la descripción del servicio. La solución propuesta se basa en una arquitectura distribuida para servicios web sensibles al contexto. El punto de partida es un escenario en el que un grupo de estudiantes universitarios desean compartir las soluciones de un ejercicio o problema estando todos ellos situados en el campus. Teniendo en cuenta los requisitos de este caso de partida, el trabajo de investigación se ha centrado en dos ámbitos, en los sistemas sensibles al contexto y en los mecanismos de descubrimiento de servicios. Dentro del primer ámbito, se ha llevado a cabo un análisis detallado de las técnicas de modelado del contexto. Las capacidades del sistema han sido ilustradas mediante un arquitectura abstracta dividida en capas. En el caso del segundo ámbito de investigación, se han descrito las soluciones de descubrimiento de servicios que tienen el soporte de las industrias. Las soluciones en las que se han integrado las características de sensibilidad al contexto junto con los mecanismos de descubrimiento han sido analizadas con una especial atención.

La solución propuesta incluye un modelado del contexto mediante el uso de ontologías, con el objetivo de describir el vocabulario de los servicios. Este modelo se comparte entre los distintos usuarios del sistema para facilitar la descripción de sus peticiones. El Devices Profile for Web Services (DPWS) ha sido integrado en la arquitectura propuesta con el objetivo de enviar, describir y descubrir los servicios Web. La metodología para la validación ha consistido en comparar distintos escenarios que usan la ontología del contexto como fuente del vocabulario, con otros en los que se usan sinónimos extraídos de Wordnet. Esta comparación se ha realizado usando los resultados de una serie de simulaciones de eventos discretos. Gracias a estos escenarios de simulación se han estudiado las métricas de rendimiento relacionadas con los mecanismos de descubrimiento, los parámetros de control y los modelos de comportamiento. Los resultados han mostrado que el uso de ontologías de contexto mejoran el ratio de descubrimiento de servicios al igual que el número medio de servicios descubiertos por petición. Escenarios con la ontología como fuente
de vocabulario han generado un overhead de los mensajes *Probe* comparando con escenarios basados en Wordnet.
Contents

1 Introduction ............................................... 1
  1.1 Motivation ............................................. 1
  1.2 Outline ................................................ 2

Part I Background

2 Context-aware systems .................................... 7
  2.1 Introduction ............................................ 7
  2.2 Context and context-awareness ......................... 7
    2.2.1 Context definitions ................................. 8
    2.2.2 Context-awareness definitions ..................... 8
  2.3 Context modeling ....................................... 9
    2.3.1 Key-value models ................................ 9
    2.3.2 Markup scheme models ............................. 9
    2.3.3 Graphical models ................................ 9
    2.3.4 Object oriented models ........................... 10
    2.3.5 Logic-based models ............................... 10
    2.3.6 Ontology-based models ............................ 10
  2.4 Context-aware system architecture ...................... 10
    2.4.1 Sensing subsystem ................................ 10
    2.4.2 Thinking subsystem ............................... 11
    2.4.3 Acting subsystem ................................. 11
  2.5 Examples of context-aware systems ..................... 11
  2.6 Summary ............................................... 14

3 Service discovery models for context-aware systems ...... 17
  3.1 Introduction .......................................... 17
  3.2 Service oriented architecture principles .............. 18
  3.3 Industry and consortia supported models for service discovery 20
    3.3.1 Jini .............................................. 20
## Contents

3.3.2 Service Location Protocol (SLP) .......................... 21  
3.3.3 Universal Plug and Play (UPnP) ......................... 23  
3.3.4 Bluetooth Service Discovery Protocol (Bluetooth SDP) . 23  
3.3.5 Web service discovery ................................ 24  
3.3.6 Semantic web service discovery ......................... 28  
3.4 Enabling context-awareness in service discovery models .... 29  
3.5 Summary ................................................ 33  

### Part II Research contributions

4 Distributed architecture for enhanced context-aware web services ........................................... 37  
4.1 Introduction ............................................ 37  
4.2 Design methodology ..................................... 38  
4.3 Context model .......................................... 40  
4.4 Service discovery mechanism ............................. 41  
4.4.1 Service matching .................................. 41  
4.4.2 Web service generation process ....................... 42  
4.5 Architecture components ............................... 43  
4.5.1 The client ........................................ 43  
4.5.2 The provider ...................................... 43  
4.5.3 The ontology service provider ....................... 44  
4.5.4 Interactions ....................................... 44  
4.6 Prototype architecture ................................. 46  
4.7 Summary ............................................... 47  

5 Validation .................................................. 49  
5.1 Introduction ............................................ 49  
5.2 Previous tasks .......................................... 50  
5.3 Validation methodology ................................. 50  
5.4 Validation via discrete-event simulation ................. 50  
5.4.1 Simulation conditions ................................ 51  
5.4.2 Validation metrics .................................. 51  
5.4.3 Simulation parameters .............................. 52  
5.4.4 User behavior models .............................. 53  
5.4.5 Simulator implementation .......................... 55  
5.5 Simulation results and discussion ....................... 55  
5.5.1 Simulation identifications ......................... 55  
5.5.2 Discovery ratio ................................... 56  
5.5.3 Mean discovered services per request ............... 58  
5.5.4 Probe messages overhead .......................... 61  
5.5.5 Results summary .................................. 62  
5.6 Summary ............................................... 63
Part III Conclusions

6 Conclusion and open problems ........................................ 67
  6.1 Thesis summary ........................................... 67
  6.2 Contributions ........................................... 68
  6.3 Future work and open problems ............................. 69

Part IV Appendix and references

A Context ontology .................................................. 73

B Web services discovery messages ............................... 77
  B.1 Web services descriptions ................................ 77
  B.2 DPWS messages ........................................... 79
    B.2.1 Hello ............................................. 79
    B.2.2 Bye ............................................. 80
    B.2.3 Probe and probe match ............................ 80
    B.2.4 Get device metadata ............................... 82
    B.2.5 Get service metadata .............................. 84
    B.2.6 Invoke .......................................... 85

C Two-State Markov-Modulated Poisson Process (MMPP2) .. 87

References .......................................................... 95

List of publications .............................................. 103
Introduction

1.1 Motivation

The interactions between users and Internet-enabled devices have evolved during the last decades. At the beginning, tasks were limited to e-mails, web interaction, and file sharing. Then, with the Web 2.0 technologies, users have become more involved in the creation of the digital content especially with services such as social networks. Nowadays, in the era of smart devices, native Internet applications, mostly running at mobile devices (iOS-based\(^1\), Android-based\(^2\), etc.) are gaining ground on browser-based ones by targeting specific services and facilitating human-computer interactions. Nevertheless, users have a passive behavior when interacting with their mobile devices. They only consume services provided by third parties. However an active behavior is also possible. It will transform users into service providers. In fact, people localized within the same ambient may share common interests and collaborate to solve problems. Let’s consider the following scenario:

Carlos is a computer science student. He is seeking for help to solve an exercise in the subject of computer architecture. His request can be achieved via different ways. He could post his petition on the University forum. He could go to the library and could search for the adequate book to solve the problem. He could also ask his classmates for help. Each of these solutions has its point of failure. In fact, the first one relies on a central service, the university forum and the communication is asynchronous. The second approach is not efficient as a matter of time. While, the last one does not assure that one of the classmates could help. So, what if he could send his request to all the university community and get help instantly, if possible and available. Therefore, Carlos takes his mobile device, accesses the “Resolve Your Problem” application, introduces the keywords that specify his request and sends his petition via the wireless network. Meanwhile, Isaac, another computer science student can

\(^2\) http://www.android.com/, Last accessed, June, 2013
offer his help for various exercises related to the computer architecture subject. Therefore, he takes his Smartphone, accesses the same application as Carlos, specifies the keywords defining the help he could offer, attaches a resource to his solution (the resource can be a book reference, a link to a web site, a meeting date or a file) and sends it via the wireless network. Two scenarios could happen. In the first one, Isaac receives Carlos’ request. Since he is providing a help resource for this petition, he will provide Carlos with the solution. In the second scenario, Carlos receives a notification from Isaac that he is offering his help. Since his is interested, he will request the resource from Isaac.

For this scenario to be successful, users have to share the vocabulary selected when describing their services or requests. Thus, the hypothesis of this thesis is that users would have better service discovery if they would share the vocabulary for their petitions. This scenario highlights two major features: Context-awareness and Service-oriented. The first one describes the environment in which the users interact. The latter specifies the way the users interact. To realize the scenario described above, we need to design a system that integrates features from the context-awareness domain and the service-oriented paradigm. First, the vocabulary has to be structured within a model that has a sharing feature. Then, this model has to be integrated with a service discovery mechanism. The resulting solution will be a distributed architecture for context-aware services. To accomplish this goal, we set the following objectives:

- The modeling of the context that describes the services to be provided within a specific ambient.
- The selection of an adequate service discovery mechanism.
- The design of a distributed architecture for context-aware services by integrating the previous targets.
- The creation of an architecture prototype.
- The definition of performance metrics that are related to the service discovery process.
- The validation of the hypothesis via a discrete-event simulation of the proposed solution.

1.2 Outline

We have divided this document into four main parts: background, research contributions, conclusions, and appendix and references.

- **Part I: Background** In the first part, we have included two surveys. The first one (chapter 2) deals with context-aware systems. We have reviewed concepts such as modeling techniques and architecture. Existing systems have been also discussed. In chapter 3, we have presented the second survey. It focuses on service discovery mechanisms. We have covered industry and consortia supported approaches as well as research initiatives.
• **Part II: Research contributions** The second part includes two chapters. In chapter 4, we have presented the design of our solution that consists in a distributed architecture for enhanced context-aware web services. By following a design methodology, we have described the architecture components, their behaviors, and their interactions. A context model for service vocabulary has been specified and integrated with a discovery mechanism. We have proposed a prototype for the resulting solution. Chapter 5 is devoted to the validation of our hypothesis. We have presented a simulation-based approach by specifying performance metrics and user behavior models. Finally, simulation results were discussed.

• **Part III: Conclusions** Chapter 6 summarizes the thesis contributions and presents future work and open problems.

• **Part IV: Appendix** This part includes the appendixes and the bibliography. First, the context ontology is listed in appendix A. Then, we have presented descriptions of the context ontology web service, an example of a solution service, and the DPWS messages in appendix B. Finally, the Two-State Markov-Modulated Poisson Process (MMPP2) distribution is detailed in appendix C.
Part I

Background
In this chapter, we cover design aspects of context-aware systems. First, various context definitions are presented. Then, different modeling techniques are discussed. After that, an abstract layered architecture is described. Finally, existing context-aware systems are analyzed according to criteria such as architecture, context model and application domain.

2.1 Introduction

With the omnipresence of wireless devices, users found themselves surrounded by an increasing amount of information. Applications designers had to follow this change and to propose new software architectures capable of dealing with multiple information sources. The paradigm of context-aware system was born. This new vision consists in characterizing the context elements and specifying the system capabilities. As a first step in the design of our solution, this chapter is devoted to context-aware systems aspects: core concepts definitions, context modeling approaches, and architecture design. The rest of this chapter is organized as follows. In section 2.2, context and context-awareness definitions are discussed. Context modeling techniques are described in section 2.3. Section 2.4 deals with the abstract layered architecture of context-aware systems. A review of existing approaches is detailed in section 2.5. Finally, we include the summary of the chapter.

2.2 Context and context-awareness

In this section two questions will be answered (i) what does context mean? (ii) when systems become context-aware?
2.2.1 Context definitions

In Merriam-Webster dictionary\(^1\), context is defined as (1) the parts of a discourse that surround a word or passage and can throw light on its meaning (2) the interrelated conditions in which something exists or occurs: environment, setting.

Cambridge dictionary\(^2\) defines context as the situation within which something exists or happens, and that can help explain it.

Dictionary’s definitions give a general significance to the context. Therefore, they cannot be extrapolated to the computing environment. Hence, researchers intended to propose their own definitions.

Schillit et al.\(^83\) defines the context within the paradigm of mobile computing: Context encompasses more than just the user’s location, because other things of interest are also mobile and changing. Context includes lighting, noise level, network connectivity, communication costs, communication bandwidth, and even the social situation. This definition is related to specific types of context. It would be difficult to apply it with other contexts.

Pascoe\(^75\) describes context as the subset of physical and conceptual states of interest to a particular entity. This definition considers the context from a single element point of view and not from the whole situation of the application and the users.

Dey\(^30, 29\) gives an operational definition of the context: 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. This definition is open to different kinds of context whether it is physical or immaterial. It focuses on the characterization of the situation resulting on the interactions between the user and the application.

Based on Shillit’s and Dey’s definitions, Chen\(^22\) deducted that context is information about a location, its environmental attributes (e.g., noise level, light intensity, temperature, and motion) and the people, devices, objects and software agents that it contains. Context may also include system capabilities, services offered and sought, the activities and tasks in which people and computing entities are engaged, and their situational roles, beliefs, and intentions.

2.2.2 Context-awareness definitions

Shillit et al.\(^83\) qualify a system with the context-aware feature if it adapts according to the location of use, the collection of nearby people, hosts, and accessible devices as well as to changes to such things over time. In this definition, the context-awareness is assimilated to an adaptability capability.

Dey\(^30, 29\) considers that a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy

\(^1\) http://www.merriam-webster.com/, Last accessed: June, 2013
\(^2\) http://dictionary.cambridge.org/, Last accessed: June, 2013
depends on the user’s task. This definition focuses on the resulting behavior from the acquired context information. It offers a user-centric view of context-awareness.

2.3 Context modeling

A context model is needed to define the structure of the context data. Strang and Linnhoff [87] surveyed the most relevant context modeling approaches.

2.3.1 Key-value models

These models represent the most simple data structure for modeling contextual information. It consists in associating context attributes with specific values. The system proposed by Schilit et al. [83] is an example of key-value model. In spite of its easiness, these models lack of capabilities for complex data structuring through the description of the associations between context concepts.

2.3.2 Markup scheme models

Markup based models are hierarchical data structure consisting in markup tags with attributes and contents. They are typically used for profiles. Some approaches based on these modeling extend profile standards such as the Composite Capabilities/Preferences Profile (CC/PP) [51] or the User Agent Profile (UAProf) [73] in order to represent a more complex contextual information. Others propose their proper XML-based descriptions of the context. Markup schemes are simple, flexible, and structured. However, they cannot handle data ambiguity. This task has to be done on the application level. Furthermore, data relationship cannot be defined through this approach.

2.3.3 Graphical models

The Unified Modeling Language (UML) is used for graphical modeling of the context. Thanks to its graphical components and to its generic structure, UML is adequate for context modeling. Brauer [7] illustrates air traffic management concepts with UML. Object-Role Modeling (ORM) is another graphical formalism that can be used for context representation. Henricksen et al. [46] proposed a contextual extension of ORM. This approach aims to derive an entity-relationship model. The graphical modeling is more suitable for human structuring purposes than for computer formalisms.
2.3.4 Object oriented models

Object oriented modeling takes advantages of any object oriented feature namely encapsulation, reusability and inheritance. Context processing details are encapsulated on an object level and hence hidden to other components. Contextual information is accessed through specified interfaces. An approach that illustrates this model is the Active Object Model of the GUIDE project [25]. Interoperability and sharing data are major issues due to the domain specific of the object oriented modeling.

2.3.5 Logic-based models

In these models, facts, expressions, and rules are used to define the context. Thus, they have a high degree of formality. However, the description of relationship between context information is not addressed. The proposition presented in [1] illustrates a model of formal context based on situation theory. The unavailability of full logic reasoners for context-aware systems constitutes an applicability issue.

2.3.6 Ontology-based models

An ontology is an explicit specification of conceptualization [35]. It represents a description of concepts, properties and relationships between concepts. According to the evaluation made by Strang and Linnhoff-Popin [87], ontology-based models are the most expressive ones. In fact, they have higher and more formal expressiveness. It is also possible to use ontology reasoning techniques. Knowledge sharing and reuse can be performed between heterogeneous context sources. Ontologies are constructed using formal languages such as Web Ontology Language (OWL) [62] or Resource Description Framework (RDF) [44]. CoBrA [23] and SOCAM [36] are examples of systems which use ontology-based modeling.

2.4 Context-aware system architecture

From an abstract point of view, a context-aware system can be divided into three basic subsystems [57]: the sensing subsystem, the thinking subsystem and the acting subsystem. The functionalities associated to each subsystem are grouped within layers [6]. Figure 2.1 illustrates the abstract layered architecture for context-aware systems.

2.4.1 Sensing subsystem

The sensing subsystem is composed of the sensors and the raw data retrieval layers. The first one is a collection of data sources that provide the system with
2.4.2 Thinking subsystem

The thinking subsystem includes the preprocessing and the storage layers. The first one treats the data retrieved by the layer below. This is fulfilled by applying reasoning techniques in order to obtain information more suitable for application designers. This layer plays the role of a data aggregator by composing different sensor sources into more abstract information. The storage layer provides the users with a structured data.

2.4.3 Acting subsystem

The application layer constitutes the acting subsystem. The system behaviors according to specific contextual information are implemented in this layer.

2.5 Examples of context-aware systems

In this section, we will discuss different context-aware systems focusing mainly on three features: the context model, the architecture and the application domain. A review of various context-aware systems surveys is presented at the end of the section. Table 2.1 summarizes the different discussed context-aware systems.
The Context toolkit [28] is a framework for the development and the deployment of context-aware services. The architecture is centralized via a discoverer where sensor units (widgets), aggregators and interpreters are registered in order to be found by client applications. The context information is handled through an XML-based attribute-value-tuples. The authors demonstrate the use of the toolkit with two applications: an active badge call-forwarding and a mobile tour guide.

The Cooltown [50] project, developed by HP Lab, focuses on allowing mobile devices to interact with a web-enabled environment. Depending on the user position, context information can be retrieved from a set of points of interest. This information is communicated via a URL and described with an XML document. The project presents two scenarios of Cooltown implementation: a museum visit and a conference room.

Gaia [79] is a CORBA-based middleware which consists of an operating system with context-aware feature. It aims to support active space applications to retrieve as well as to publish contextual information. This information is represented using first-order predicate with arguments. To demonstrate Gaia functionalities, a presentation manager for a meeting scenario was developed.

In [48], the authors presented a generic framework that facilitates the development and deployment of context-aware adaptable web services. Context information related to consumers and their environment is used to provide customized web services. This information is transmitted as an XML document inserted into the SOAP header. The framework demonstration consists of a mobile application which interacts with the amazon web service in order to retrieve personalized information according to the client context.

CoBrA [22] is an agent-based architecture for supporting context-aware systems in smart spaces. The core of CoBrA is an intelligent context broker that manages a shared contextual model on the behalf of community of agents. The contextual information is represented with ontology. It describes the domain in which devices, web services and people interact. An intelligent meeting scenario [21] was developed as an example of the CoBrA system.

SOCAM [36] is a service-oriented context-aware middleware architecture for the building and rapid prototyping of context-aware services. Context information is acquired through distributed providers and transformed to an ontology-based model. This information is processed by a centralized interpreter. It is used as an input for context-aware web services in order to adapt their behaviors. Home-domain and vehicle-domain ontologies were designed as SOCAM applications.

CoWSAMI [3] is a service-oriented middleware that supports context-awareness in pervasive environments. The architecture uses web services as interfaces to context sources. The contextual information is represented with an extension of the Web Service Description Language (WSDL). A smart vehicle example (CyberCars) illustrates the functionalities of CoWSAMI.
The Anyserver platform [42] is a client-proxy-server architecture supporting various types of context sources such as device information, networks, and application type. Contextual information is represented in XML. An image management system was implemented as an application for the AnyServer platform.

CAMUS [68] is a context-aware middleware for ubiquitous computing systems. It supports heterogeneous and distributed sensing agents. The contextual information related to a home domain is described with an OWL-based ontology.

CA-SOA [24] is a context-aware service oriented architecture. It is composed of an agent platform, a centralized service repository, and a semantic matchmaker. The context-awareness feature is used to enhance the service discovery mechanism with a semantic description of the services and the requests. A business meeting scenario is presented as a case of application of CA-SOA.

<table>
<thead>
<tr>
<th>CA systems</th>
<th>Context model</th>
<th>Architecture</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cooltown [50]</td>
<td>Markup scheme</td>
<td>Web-based</td>
<td>Museum, conference room</td>
</tr>
<tr>
<td>Gaia [79]</td>
<td>4-ary predicates</td>
<td>CORBA-based Middleware</td>
<td>Presentation manager</td>
</tr>
<tr>
<td>Keidl [48]</td>
<td>Markup scheme</td>
<td>Web service-based framework</td>
<td>Book store information service</td>
</tr>
<tr>
<td>CoBrA [22]</td>
<td>Ontology-based</td>
<td>Agent-based</td>
<td>Meeting room system</td>
</tr>
<tr>
<td>SOCAM [36]</td>
<td>Ontology-based</td>
<td>Distributed with central context interpreter</td>
<td>Smart home, smart vehicle</td>
</tr>
<tr>
<td>CoWSAMI [3]</td>
<td>Markup scheme</td>
<td>Web service-based infrastructure</td>
<td>Smart vehicle</td>
</tr>
<tr>
<td>Anyserver [42]</td>
<td>Markup scheme</td>
<td>client-proxy-server model</td>
<td>Image management system</td>
</tr>
<tr>
<td>CAMUS [68]</td>
<td>Ontology-based</td>
<td>middleware framework</td>
<td>Smart home</td>
</tr>
<tr>
<td>CA-SOA [24]</td>
<td>Ontology-based</td>
<td>Centralized service architecture</td>
<td>Business meeting</td>
</tr>
</tbody>
</table>
Context-aware systems are subjects to various surveying works. Baldauf et al. [6] discussed the design principles for context aware-systems including architecture and context models. The survey discussed different frameworks and systems that include Gaia [79], Hydrogen [47], CASS [32], CoBrA [22], Context Toolkit [28], CORTEX [86], SOCAM [36] and Context Management Framework [54].

In [91] various context-aware web service systems are compared. The survey focused on analyzing the context modeling, the context sensing, the distribution, the security and privacy, and the adaptation techniques. The authors examined existing systems such as CA-SOA [24], CoWSAMI [3], Keidl approach [48], the Anyserver platform [42], the inContext project [92], and the Akimo project [74].

Chen et al. [20] surveyed context-aware applications in mobile computing. The authors focused on discussing the sensing and the modeling of the contextual information. Security and privacy issues were also considered.

In [85], various context-aware frameworks were compared. The study includes approaches such as: Context Toolkit [28], CoBrA [22], SOCAM [36], CMF [54]. The authors also presented their own approach of context-ware framework (STU21). The discussion was based on criteria such as: software architecture, context representation, intelligence, and application domains.

Martin [60] studied the relationship between web service and context. The survey covers cases such as: Task computing3, MyCampus [81], OWL-SF [65], Semantic Discovery Service [59], and ConWes [82]. The author also discussed various challenges in representing and reasoning about context for services.

In [52], various context-aware middleware systems were reviewed. The survey includes: Aura [33], CARMEN [8], CARISMA [16], Cooltown [50], CORTEX [86], Gaia [79], MiddleWhere [78], MobilPADS [19], and SOCAM [36]. Systems capabilities were categorized according to a specific taxonomy. This categorization includes environment, storage, reflection, quality, composition, migration, and adaptation.

Bolchini et al. [11] evaluated different context models via a defined framework. They examined parameters related to context dimension (time, space, user profile, etc), types of formalism, and context management features (construction, reasoning, monitoring, etc.). The survey compares different context-aware approaches such as: SOCAM [36], CoBrA [22], CASS [32], etc.

2.6 Summary

In this chapter, we discussed various aspects related to context-aware systems. First, we defined basic concepts such as context and context-awareness. Dey's approach gives an open and operational view to the context and a user-centric one regarding the context-awareness. A review of modeling techniques showed

---

3 http://www.taskcomputing.org/, Last accessed: June 2013
that the ontology-based has a higher degree of expressiveness as well as a sharing capability. After that, we described system architecture as a layered one divided into three subsystems: sensing, thinking and acting. Finally, different existing context-aware systems were compared according to their context model, architecture, and application domain.
In this chapter, we present the state of the art on service discovery models. The scope of this work falls under the service oriented paradigm. The main concepts of this framework are described with a focus on service discovery feature. Industry and consortia have been supporting various mechanisms of service discovery. This chapter includes a survey on these standards and specifications. Researchers have also been interested in designing such models especially in adding context-awareness to the discovery process. Various research initiatives are reviewed at the end of this chapter.

3.1 Introduction

The purpose of context-aware (CA) systems is to propose services within a heterogeneous ambient and via devices with special requirements. A design approach for these systems is to consider them as a Service Oriented Architecture (SOA). In fact, interactions among the components include announcing, searching and invoking services. On one hand, these interactions fall under the SOA vision. On the other hand, they have to stick to specific system requirements. The devices limitations and the context in which the components interact are two major constraints. The expected result from a SOA is the invocation of the adequate service to a specific request. This process is simple if the requester and the service provider know each other at run time. But, due to the heterogeneity of current computers and especially CA systems, the task of finding the appropriate service is more complex. This mechanism is called service discovery (SD). Several models of service discovery have been specified for SOA. Some of them may be suitable for CA system than others. A study of the existing techniques for SD will help to establish the appropriateness of existing SD models for CA systems. Jini, Service Location Protocol (SLP), Bluetooth discovery model and Universal Plug and Play (UPnP) constitute a good set of model to explore. Web service specifications (WS-*) include mechanisms for web service discovery such as Universal Description Discovery and
Integration (UDDI) and Devices Profile for Web Service (DPWS). Introducing semantic in service oriented paradigm via the use of ontology has opened a new branch of discovery models. Approaches such as Semantic Markup for Web Services (OWL-S) and Web service modeling ontology (WSMO) are described as part of this new type of model. Since these models are not mainly oriented to CA systems, some investigation works proposed their enhancements. Others defined their own solutions for SD in context-aware systems.

The rest of this chapter is organized as follows. In section 3.2 service oriented concepts are described with a focus on the service discovery design principles. Standards and specifications for service discovery are reviewed in section 3.3. In section 3.4 different research initiatives dealing with context-awareness in service discovery are surveyed. Finally, we include the summary of the chapter.

3.2 Service oriented architecture principles

A service oriented architecture (SOA) is a software architecture in that loosely-coupled services are defined using a description language and have invocable interfaces that are called to perform processes. From a conceptual level, a SOA is composed of three core parts:

- The service provider: It defines the service description and publishes it to the service registry.
- The service requester also called service client: It accesses to the register directory to find service description and their providers.
- The service registry/directory: It is an intermediate between the service provider and the service registry. It implements a matching mechanism that is responsible of finding the adequate service for the corresponding request.

The activities that can be performed in a SOA as described in figure 3.1 are: publishing, discovering and invoking. First, the service provider has to publish the service description. Then, the service requester queries the registry in order to find the adequate service description. Finally, the requester invokes the desired service and interacts with the provider in order to perform the service.

Designing a service discovery relies on four main axes: service description, service selection, registry structure, and communication mechanism.

- The service description is essential when publishing a service. The available methods for this task vary according to the degree of expressiveness: key/value, template-based and semantic description. In the key/value approach, services are characterized by a set of key/value pairs. Requesting a service consists in specifying the exact value of an attribute. In case there is a query language implemented, the attributes values can be compared to the requested ones using operators. The template-based description
is done via semi-structured languages such as eXtensible Markup Language (XML). The semantic description relies on the use of ontologies. They have higher and more formal expressiveness. It is also possible to use ontology reasoning techniques. Knowledge sharing and reuse can be performed between heterogeneous context sources.

- The service selection process relies on a matchmaking algorithm depending on the degree of expressiveness of the service/request. The matchmaking process compares the discovery request generated by the client with the service advertisement announced via the provider. It relies on the information retrieval paradigm. The more expressive the retrieval approach is, the more the matching algorithm is complex and operates slowly. Klein [53] distinguishes four service retrieval approaches:
  - Keyword-based retrieval: It uses keyword to perform search from the service request. This approach is not highly expressive in order to capture the semantics of a request.
  - Tables-based retrieval: Services and requests are represented as tables with attribute-value pairs and then matched.
  - Concept-based retrieval: It defines ontologies for organizing services. The retrieval is type-based rather than keyword-based.
  - Deductive retrieval: Service semantics are expressed formally using logic. The retrieval is done via deducing the adequate service for the functionality described in the query.

- The discovery process depends on the service registry architecture: centralized or distributed. From one hand, the centralized-based model implies a dedicated directory that maintains service information and processes requests. On the other hand, the distributed model means that any component of the system maintains a local part of the registry. The service requests are distributed across the registries transparently to the
clients. There is a service-oriented model that does not include a registry: a directory-less model. In the latter model, the provider has to announce to periodically its services via multicast protocols. The client has to probe in order to search for the adequate service to its request.

- Another aspect of service discovery design is the communication mechanism for retrieving services. We can find two approaches: Query/pull and Notification/push. The first one implies that the clients query for services. In the second approach, requesters can register and wait to be notified if an adequate service is published.

### 3.3 Industry and consortia supported models for service discovery

In this section, standards and specifications for service discovery are discussed. Figure 3.2 illustrates the major SD approaches.

![Standards and specifications for service discovery](image)

**Fig. 3.2.** Standards and specifications for service discovery

#### 3.3.1 Jini

Jini also called Apache River [89], 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-centered. Since it is SOA-based, Jini’s key concepts reflect the basic principles of service computing. The lookup service (LUS) plays the role of registry. Service provider (Jini service) and service requester (Jini client) find each other via the LUS. The core of Jini communication is a trio of protocols namely discovery, join and lookup. Interactions among the different Jini architecture components are as follows: When a Jini service or client starts up; it sends a multicast request to
3.3 Industry and consortia supported models for service discovery

search for a lookup service. Once one or more LUSs respond to the request, the provider starts the join process. The Jini service registers a service object (proxy) and its attributes with the lookup service. The object consists of a Java interface for the service with the corresponding invocation methods. The Jini client requests a service to the lookup service. Then, the LUS sends a copy of the service object to the client. Finally, the requester uses the proxy to communicate directly with the service provider. The lookup service sends at startup and periodically a multicast announcement via User Datagram Protocol (UDP) to the network components. Figure 3.3 shows the various communication scenarios that describe the Jini discovery process.

![Fig. 3.3. Service discovery interactions in Jini](image)

3.3.2 Service Location Protocol (SLP)

The Service Location Protocol (SLP) [40] is being specified by the IETF. It provides a scalable framework for the discovery and selection of network services. SLP architecture includes three main components:

- **User Agent (UA):** It plays the role of a proxy for the client in the discovery process.
- **Service Agent (SA):** It announces the location and attributes of the service.
- **Directory Agent (DA):** It registers services published by the SAs in its database and responds to service requests from UAs.

To accomplish their respective roles UA and SA have to discover a DA. There are three methods for DA discovery: static, active and passive. In the static approach, SLP agents retrieve the address of DA via Dynamic Host Configuration Protocol (DHCP). In the active method UAs and SAs use SLP multicast
group address to send service requests. Then, the DAs that are listening on this address respond via unicast to the agent. In passive discovery, DAs announce periodically their services via multicast. Hence, user and service agents are able to know the DA address and to communicate directly with it. These discovery scenarios are more adequate for large networks with many services. In a small network, the discovery process can be fulfilled without the DA. In this case, UAs send periodically their service requests to the SLP multicast address. The SAs announcing the service will send a unicast response to the UA. Moreover, SAs announce their presences via multicast. Figure 3.4 shows the different communication scenarios in SLP discovery process.

Fig. 3.4. SLP service discovery scenarios
3.3 Industry and consortia supported models for service discovery

3.3.3 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) [93] 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 specifications defining the addressing and the discovery of resources as well as the description and the control of the services within the network. The UPnP architecture includes two main components: devices and control point. Devices offer services over the network whereas control points consume these services. The UPnP discovery process is based on the Simple Service Discovery Protocol (SSDP). The process is directory-less. When a device joins the network, it announces its service to the control points. It sends HTTP requests over multicast via UDP namely HTTPMulticast. These announcements contain information about the service types and links to the descriptions. SSDP permits to a new control point to look for devices via multicast. The information received by the control point from the device during the discovery phase do not allows it to invoke a service. To do so, the control point must retrieve the device’s description from the link provided previously. This description is XML-based and includes a list of the provided services and URLs for control, eventing and presentation. Once the description retrieved, the invocation process is initiated using SOAP (Simple Object Access Protocol) messages. Figure 3.5 show the interactions during the discovery process.

![UPnP service discovery interactions](image)

Fig. 3.5. UPnP service discovery interactions

3.3.4 Bluetooth Service Discovery Protocol (Bluetooth SDP)

Bluetooth comes with its own protocol stack. As part of it, it offers its proper service discovery methods: the Bluetooth Service Discovery Protocol [10]
Service discovery models for context-aware systems

(Bluetooth SDP). The SDP specifies the behavior of a Bluetooth client application in order to discover the available services in the Bluetooth servers. The protocol also establishes the searching mechanism for services. Service discovery in Bluetooth relies on a request/response model (figure 3.6). This model allows devices to discover Bluetooth services offered within the vicinity via two modes: searching and browsing. Searching interaction consists in retrieving a specific service, while browsing is the process of looking for the available services. These inquiry methods are possible only if the devices are first discovered then linked. Therefore, in the discovery process, the devices assume specific roles: Local Device (LocDev) and Remote device (RemDev). A LocDev initiates the service discovery mechanism. This device implements the client part of the Bluetooth SDP architecture. The service discovery application (SRVDscApp) is used by a user to start discovering other devices. The RemDev is the device that responds to the requests of the LocDev. It contains the server part of the Bluetooth SDP. Answering the client inquiries is done by consulting the service records database.

![SDP client-server interaction](Source: Figure 2.1 [10] p.203)

3.3.5 Web service discovery

In order to explore the web service discovery domain, a definition of web service has to be adopted. According to the W3C [41]: "A web service (WS) is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards". From this definition, two major specifications arise:

- Web Service Description Language (WSDL): "WSDL is an XML format for describing network services as a set of endpoints operating on messages"
containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint." [26]

A WSDL document (figure 3.7) describes a web service using the following elements:

- Types: A container for data type definitions using some type system such as XSD (XML Schema Definition).
- Message: An abstract, typed definition of the data being communicated.
- Operation: An abstract description of an action supported by the service.
- Port Type: An abstract set of operations supported by one or more endpoints.
- Binding: A concrete protocol and data format specification for a particular port type.
- Port: A single endpoint defined as a combination of a binding and a network address.
- Service: A collection of related endpoints.

Fig. 3.7. WSDL 1.1 document structure

- Simple Object Access Protocol (SOAP): "SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible
messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.” [38]

WS discovery models rely mainly on the specifications described above. In the rest of this section, these models are described.

**Universal Description Discovery and Integration (UDDI)**

Universal Description Discovery and Integration (UDDI) [69] is a directory-based web service discovery mechanism. It is an open industry initiative, sponsored by the Organization for the Advancement of Structured Information Standards (OASIS). UDDI specifies a framework for describing and discovering web services. The discovery model is centralized. It consists of a business registry that operates as a naming and directory service. The service provider publishes its services to this registry and the service client discovers services by requesting the UDDI registry. This mechanism is fulfilled through two main functionalities of UDDI:

- UDDI defines data structures and application programming interfaces (APIs) for publishing service descriptions.
- UDDI enables the user to query the registry in order to look for published descriptions.

The UDDI registry is organized in three categories:

- The white pages contain information related to organizations such as contact information and the provided services.
- The yellow pages store industrial categorizations based on standards or user defined taxonomy.
- The green pages contain technical information on how a web service can be invoked such as WSDL.

The discovery process is done via APIs. UDDI defines three types of users that can access the APIs: service providers that publish services, service requesters that query for services, and other registries that exchange information on services. The publisher and the inquiry APIs are the most relevant for the service discovery. The first one includes operations for service announcements that can be used by service providers. The latter API includes operations to locate registry entries and details about a specific one.

**Devices Profile for Web Services (DPWS)**

The Devices Profile for Web Service (DPWS) [72] 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.

The DPWS protocol stack is illustrated in figure 3.8. It includes the following WS-* specifications:

- **WS-Addressing [13]**: It describes addressing information in SOAP message header independently from the transport protocol.
- **WS-Security [70]**: It allows secure communication to web services by providing mechanisms of signing, encryption and identity authentication.
- **WS-Policy [95]**: It allows services to express their policies (security, quality of service) and clients to specify their needs.
- **WS-MetadataExchange [58]**: It specifies metadata description of hosted services and hosting devices.
- **WS-Eventing [12]**: It defines a protocol of event notifications via a subscription mechanism.
- **WS-Discovery [71]**: 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. In Appendix B, matching rules are detailed.

WS-discovery defines two operational modes: an ad-hoc mode and a managed mode. The first one consists of a distributed communication model. Clients and devices communicate 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 a 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 latter 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 metadata exchange phase in order to get the corresponding metadata of the device (6,7) and the searched service (8,9). This is fulfilled via the WS-MetadataExchange and WS-Transfer [2]. Once the client has the metadata, it can invoke the desired service (10,11). When a device leaves the network, it sends a multicast Bye message (12). Figure 3.9 shows the message exchanges during the service discovery phase in DPWS.

3.3.6 Semantic web service discovery

Semantic Web Service Discovery (SWSD) is another subtopic of service discovery. It introduces the use of semantic web technologies in the process of service discovery. The discovery mechanism relies on the semantic service description via ontologies and semantic service matchmaking.

Major approaches for semantic service description are briefly revised as follows:

- Semantic Markup for Web Services (OWL-S) [61]: OWL-S (formerly known as DAML-S) is an ontology of services that enables users and software agents to discover, invoke, compose and monitor web resources in an automatic way. The automatic web service discovery refers to the automated process for location of web services taking into account some client requirements. In order to perform this process, first, the information needed for the discovery is specified with a computer-interpretable semantic markup (ontology) rather that human-interpretable (web search request). Then a service registry or an ontology-enhanced search engine locates the adequate service automatically using the reasoning capabilities of the semantic description.
3.4 Enabling context-awareness in service discovery models

The models supported by industries and consortia are not initially specified for CA systems. In light of this, some research teams described enhancements for these models. Others proposed new solutions adapted for CA systems. In this section, we discuss existing research initiatives of service discovery for CA systems. Table 3.1 summarizes the discussed research initiatives for service discovery.

In [90], an enhancement of UPnP discovery mechanism is presented. It consists of an ontology-based representation for UPnP devices and services. The approach binds UPnP devices to abstract service descriptions. It also provides ontology-based device selection, service invocation and context-aware adaptation. Thus, the communication model as well as the service architecture is the same as UPnP. The amelioration is at the service description and selection level.

- Web Service Modeling Ontology (WSMO) [27]: It provides a conceptual framework for semantically describing web services and their specific properties. It supports different approaches for web service discovery [49]: Keyword-based, simple description-based and rich semantic description-based.

Fig. 3.9. DPWS message exchanges
U-CAFÉ [56] is a ubiquitous and context-aware computing framework for collaborative engineering. It uses web services and Jini services to support engineering service federation and seamless interactions among persons, devices, and different types of engineering services. Web services are used for back-end communications, whereas Jini services for front-end communications. The framework includes both UDDI-based and Jini discovery mechanisms. The context feature is operated via a broker that facilitates reasoning and querying of contexts.

In [15], Broens presents a context-aware and ontology-based service discovery approach. Ontologies describe user requests, service properties and contextual information. These semantic descriptions are used in the matching process. It aims to classify the services according to the requests into matching types.

The Group-based Service discovery Protocol (GSD) [18] is a discovery protocol for mobile ad-hoc networks. It describes a mechanism of request forwarding based on a group classification of services. It also includes a peer-to-peer caching process for service announcements. Services are described using the Web Ontology Language (OWL). The services are organized into groups following a hierarchy. Service providers periodically announce their service lists. The receiver nodes forward these announcements to the others nodes within the radio range. The message forwarding is limited by a maximum number of hops. Each node stores the service advertisement in its service cache along with its local services. GSD allows nodes to announce the group information of services seen in the vicinity. Apart from service advertisements, GSD specifies the service requesting process. First, the service request is matched with the node local cache. In case of cache miss, a selective forwarding of the request is initiated. Thus, the message is transmitted to nodes associated with the group of the requested service. The service description and selection are semantic-based. The approach is directory-less because it is oriented for mobile ad-hoc networks. GSD communication mechanism supports both query/pull and notification/push models.

Avancha et al. [4] enhanced service discovery in Bluetooth with semantics. In regular Bluetooth SDP, services are associated with UUID (Unique Universal Identifier). The new matching mechanism is improved via the use of semantic information associated with services. An ontology and a Prolog-based reasoning engine are introduced for more efficient discovery. The service description and selection are semantic-based. The registry structure and the communication mechanism have no changes compared to Bluetooth SDP.

The CA systems presented below were described in section 2.5 from the point of view of the architecture and the functionalities. As follows, their service discovery features are highlighted.

Context toolkit [28] discovery mechanism is centralized. It uses a single registry called discoverer. When started, all the components register with the discoverer at a known network address and port. The discoverer pings each registered component at a pre-specified frequency to ensure that each
3.4 Enabling context-awareness in service discovery models

Cooltown [50] discovery mechanism is location-aware. Depending on the user position, a set of points of interest (i.e. people, places, things) is retrieved. Each point is associated with a web page describing its context and discoverable via its URL or ID. A beacon-based system is used in order to sense the location of these points of interest.

SOCAM service discovery mechanism [37] operates thanks to the semantic description and matching for services and to the service locating service. The latter allows context providers and the context interpreter to advertise their presences. It also enables users or applications to locate these services. It is capable of tracking changes of context providers. It also implements different matching algorithms. Although the SOCAM middleware is a distributed architecture, the discovery mechanism is centralized through the service locating service.

CoWSAMI [3] provides a dynamic and scalable web service discovery mechanism. The service descriptions are specified using WSDL. The message exchanged follows the SOAP standards. The discovery mechanism relies on two core components: the Naming & Discovery service and the Context Manager. The first one maintains two repositories by storing information related to web services that are present within the environment. The discovery mechanism is performed using URIs instead of XML documents. The context manager is responsible for categorizing the services registered within the Naming & Discovery service. This task is realized according to the context information that characterizes each service.

The Anyserver platform [42] discovery mechanism is UPnP-based. The process is realized via the Anyserver component. It consists of a server that provides integrated services such as short message services, image uploading and downloading, and on-line audio and video service. It also manages the context information representing the user profiles. This context data has an XML-based representation.

CAMUS [43] service discovery mechanism relies on the Jini lookup service. A CAMUS service is described with an ontology. Once a service registers with Jini LUS, a notification is sent to the service delivery manager. The latter associates each service with the corresponding ontology, and stores it in the service ontology repository. Service requesters query the delivery manager to search for the adequate service. Compared to the Jini mechanism, CAMUS novelty consists in associating ontology with services. Despite the architecture is distributed, the discovery process is centralized via the delivery manager.

CA-SOA [24] discovery mechanism is composed of two phases: capability matching and context matching. The first one uses DAML_S/UDDI methodology for matching requests with web services. In the second phase, a context matching is performed in order to refine the query. A broker agent is used to handle the interactions between the service agent, the request agent and the service repository. From one hand, the broker helps the service agent to
publish its services to the service repository. On the other hand, the request agent uses the broker in order to forward the request to the service planner and the context reasoner responsible for the two phase's discovery process. The service description and selection are semantic and syntactic by combining semantic context information with service description for UDDI. The directory structure is centralized via an UDDI repository.

<table>
<thead>
<tr>
<th>SD Approach</th>
<th>Service description and selection</th>
<th>Directory structure</th>
<th>Communication mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enhanced UPnP [90]</td>
<td>Ontology description of UPnP devices and service</td>
<td>Directory-less (UPnP-based)</td>
<td>UPnP-based</td>
</tr>
<tr>
<td>U-CAPÉ [56]</td>
<td>WSDL and Jini</td>
<td>Mixed structure</td>
<td>Jini and UDDI based</td>
</tr>
<tr>
<td>GSD [18]</td>
<td>Ontology-based</td>
<td>Directory-less</td>
<td>Query/pull and Notification/push</td>
</tr>
<tr>
<td>CAMUS [43]</td>
<td>Ontology-enhanced</td>
<td>Centralized via delivery manager</td>
<td>Jini-based</td>
</tr>
<tr>
<td>GSD [18]</td>
<td>Ontology-based</td>
<td>Centralized via the service locating service</td>
<td>Query/pull</td>
</tr>
<tr>
<td>CoWSAMI [3]</td>
<td>Ontology (OWL-S) and WSDL</td>
<td>UDDI central registry</td>
<td>UDDI-based</td>
</tr>
<tr>
<td>CoWSAMI [3]</td>
<td>WSDL and XML description of context</td>
<td>Centralized (Naming &amp; Discovery service and Context Manager)</td>
<td>SLP and Query/pull</td>
</tr>
<tr>
<td>Anyserver [42]</td>
<td>XML and UPnP representation</td>
<td>Centralized via Anyserver</td>
<td>UPnP-based</td>
</tr>
<tr>
<td>Cooltown [50]</td>
<td>Web page description</td>
<td>Centralized</td>
<td>query/pull</td>
</tr>
<tr>
<td>Context toolkit [28]</td>
<td>N/A</td>
<td>Registry-based (discoverer)</td>
<td>Yellow and white pages lookup mechanism</td>
</tr>
</tbody>
</table>
Due to their relevance in service oriented paradigm, service discovery mechanisms are subjects to various surveying works.

In [45], Helal et al. reviewed the different standards for service discovery and delivery including Jini, UPnP, Salutation, SLP and Bluetooth SDP.

The survey in [63] focused on comparing various mechanisms and frameworks according to a set of criteria. The evaluation discusses the type (fixed, wireless or constraint), the architecture (client-server, peer-to-peer or hybrid), the scale (local, enterprise or Internet), and the searching mechanism (unstructured, uninformed, informed, structured or hybrid).

Su et al. [88] surveyed the discovery protocols for ad-hoc networks. The study compares characteristics related to the directory structure, the communication mechanism (pull/push), and service features such as description, access and caching.

Rambold et al. [77] evaluated different categories of service discovery approaches. The methodology calculates a score taking into account various criterion such as description, reasoning, scalability, robustness, composition and QoS.

In [80], Rong et al. analyzed the different matchmaking models for web services as well as approaches that include context-aware features.

Surveys [67] and [66] focused on web service discovery mechanism. In [55], Kunster et al. presented an evaluation of semantic service discovery research initiatives.

3.5 Summary

This chapter presented the background on service discovery. This study covered the definition of the service oriented paradigm as well as its main features namely: requesting, providing and discovering. A special interest was given to the latter one due to the nature and the goals of this document. Therefore, characteristics involved in the discovery process were detailed: service description and selection, service directory structure and communication mechanism. Specifications such as Jini, UPnP, SLP and Bluetooth SDP were detailed. When a service relies on WS-* especially on WSDL and SOAP, we are referring to web services. Approaches such as UDDI and DPWS illustrate discovery models for his category of service. If semantic is introduced in the process of description and matching, we are within the scope of semantic web service. OWL-S and WSMO are examples of approaches for semantic web service discovery. In order to relate this current work with the concept of context-awareness presented in the previous chapter, we were interested in reviewing research initiatives that enable this concept in the discovery process. Some works added the context feature by enhancing the service description and selection via ontologies whereas others included in their architecture elements capable of associating context information with their corresponding services.
Part II

Research contributions
Distributed architecture for enhanced context-aware web services

Our proposed solution is described in this chapter. We fulfilled a design methodology that leads us to build a prototype of the service architecture. First, the context model for service is described via ontology. Then, the DPWS discovery mechanism is integrated within our proposal. After that, we describe the activities of the components: client, provider and context ontology provider. Finally, we illustrate the prototype elements.

4.1 Introduction

The objective of this chapter is to propose a service oriented architecture with a context-aware feature. In this architecture, we will focus on the discovery mechanism as well as the context in which interact the users. From an abstract point of view, our solution deals with the thinking and the acting subsystems discussed in section 2.4 i.e the preprocessing, storage and application layers. The sensing subsystem is out of the scope of this research.

The starting point in the design of our solution is the motivation scenario described in section 1. The scenario is as follows:

Carlos is a computer science student. He is seeking for help to solve an exercise in the subject of computer architecture. His request can be achieved via different ways. He could post his petition on the University forum. He could go to the library and could search for the adequate book to solve the problem. He could also ask his classmates for help. Each of these solutions has its point of failure. In fact, the first one relies on a central service, the university forum and the communication is asynchronous. The second approach is not efficient as a matter of time. While, the last one does not assure that one of the classmates could help. So, what if he could send his request to all the university community and get help instantly, if possible and available. Therefore, Carlos takes his mobile device, accesses the “Resolve Your Problem” application, introduces the keywords that specify his request and sends his petition via the wireless network. Meanwhile, Isaac, another computer science student
can offer his help for various exercises related to the computer architecture subject. Therefore, he takes his Smartphone, accesses the same application as Carlos, specifies the keywords defining the help he could offer, attaches a resource to his solution (the resource can be a book reference, a link to a web site, a meeting date or a file) and sends it via the wireless network. Two scenarios could happen. In the first one, Isaac receives Carlos’ request. Since he is providing a help resource for this petition, he will provide Carlos with the solution. In the second scenario, Carlos receives a notification from Isaac that he is offering his help. Since his is interested, he will request the resource from Isaac.

Services provided within the campus ambient include academic, administrative and leisure services. The scenario can be extrapolated to another environment such as an airport or a shopping center.

Three issues arise from this scenario:

- Since each user will describe its service or request with its own words, there will be a problem in matching the petitions. One solution is to install an efficient matching algorithm in the users’ devices. The more complex the algorithm is the more resources it consumes. Since we are dealing with resource-constraint devices, we cannot dispose of an efficient algorithm. Another solution is to establish a list of possible keywords and share it to the users. These keywords represent the context of the services offered within the campus ambient. What model is the most adequate for this shared context?
- What should be the architecture structure (distributed, centralized or decentralized)? Which discovery mechanism is the most adequate for this architecture and the users’ devices?
- How each component should behave in order to accomplish the described scenario?

To answer these questions, a design methodology should be adopted. This process will lead us to specify the context model, to describe the behavior of the different components and to choose the most adequate discovery mechanism.

The rest of this chapter is organized as follows. First, the design methodology is presented in section 4.2. It specifies the requirements needed for the design of our solution. Then, we describe the ontology-based model for the service context in section 4.3. After that, the discovery mechanism and the service generation process are exposed in section 4.4. Section 4.5 details the activities of the different architecture components. The solution prototype is illustrated in section 4.6. Finally, we include the summary of the chapter.

4.2 Design methodology

The design methodology begins with the specification of the architecture requirements by analyzing the motivation scenario. After that, we will define
each components as well as its corresponding behavior. The next step consists in choosing the adequate context model from the different approaches discussed in chapter 2.3. The service discovery mechanism will be selected from the solutions described in chapter 3. After these latter three steps, a phase of adaptation and integration is needed. The last step is to propose a prototype of the future solution. Figure 4.1 illustrates the design methodology steps.

The analysis of the motivation scenario highlights the following characteristics:

- The services are deployed in a distributed and heterogeneous environment. The communication mechanism for service discovery has to support this feature.
- The devices are mobile and resource-constraint.
- The services are domain-related. The context model has not only to reflect this characteristic but also to allow the extrapolation to other domains.

**Fig. 4.1.** Design methodology for the distributed architecture
4.3 Context model

In order to fulfill the architecture requirement concerning the context model and by considering the different modeling approaches discussed in 2.3, the ontology model is the most adequate one for our solution. Figure 4.2 represents the context ontology with instances related to the campus domain. A user has a role which can be a provider or a requester. In our scenario, user can be instantiated to staff member, a professor or a student. He or she expresses a petition which can be a problem in case the user is a requester or a solution if he is a provider. A petition is specified via different properties. It has a field (Biology, Chemistry or Computer). The field concept has a category (Administration, Course or Research). The category concept has a type (Analysis, Demonstration, Exercise, Publication, etc.). In case the petition is a solution, it has a resource (Book, File or Link). Appendix A lists the context ontology (CO) in OWL. When dealing with another environment such as an airport or a shopping mall, the ontology concepts will be instantiated according to the domain characteristics.

Fig. 4.2. Ontology-based context vocabulary model
4.4 Service discovery mechanism

The service discovery mechanism has to achieve the architecture requirements. It has to offer a distributed communication model and to be suitable for mobile devices. The survey presented in chapter 3 states that the DPWS stack protocol is adequate for our solution. In fact, its discovery mechanism is multicast-based, avoiding a central service repository. It also enables web services for resource-constraint devices. Besides, DPWS is based on a set of WS standards.

4.4.1 Service matching

A service is identified by its uniform resource identifier (URI) in an XML document. This identifier represents the service’s namespace [14]. Since URI references can be long and may contain prohibited characters for elements naming, qualified names (QNames) [94] were introduced in XML namespaces. They are used to create a mapping between the URI and a namespace prefix. Listing 4.1 shows an example of a service QName.

Listing 4.1. Service QName example

```xml
<dpws:Types xmlns:i187= "http://acsic.uib.es/solutionservice"> 
i187:ComputerPracticalExcercice1 
</dpws:Types>
```

The prefix `i187` is declared to be associated with the URI: `http://acsic.uib.es/solutionservice`. This prefix can further be used as an abbreviation for this namespace. The tag `i187:ComputerPracticalExcercice1` is a valid QName. It uses `i187` as namespace and `ComputerPracticalExcercice1` as local part.

Service matching is based on WS-Discovery specification. In [71], it is stated the following:

A Probe includes zero, one, or two constraints on matching Target Services: a set of Types and/or a set of Scopes. A Probe Match MUST include a Target Service if and only if all of the Types and all of the Scopes in the Probe match the Target Service.

A Type $T_1$ in a Probe matches Type $T_2$ of a Target Service if the QNames match. Specifically, $T_1$ matches $T_2$ if all of the following are true:

- The namespace of $T_1$ and $T_2$ are the same.
- The local name/part of $T_1$ and $T_2$ are the same.

When searching for a service, the client has to specify a QName that includes a namespace and a local part. The request is matched with a service if and only if the service QName matches the request QName.
4.4.2 Web service generation process

The generation process is used by the client and the provider. The first one uses it when defining the local part of the request service. The latter uses this process when creating a new web service. To simplify this mechanism, we suppose that the namespace is common to all the users: clients and providers. It is extracted from the context ontology.

The shared ontology will assure that clients and providers will specify the same vocabulary when creating a request or a web service. Therefore, the local part of QNames representing the service and the request will match.

After locating the context ontology, the user will select the instances that describe its request or service. Figure 4.3 describes an example of the generation of the local part. In this example, the user selects “Computing” for “Field”, “Practical” for “Category”, “Exercise1” for “Type” and “Link” and “Meeting” for “Resource”. By concatenation of the selected instances, the local name obtained will be “ComputerPracticalExercise1” and the service operations will be “getMeeting” and “getLink”.

![Fig. 4.3. Web service generation process](image)
4.5 Architecture components

The architecture is composed of three types of users: client, provider and ontology service provider. In this section, we will detail the activities of each one of them.

4.5.1 The client

The client integrates a DPWS client. First, it searches for the context ontology provider. Once found, it uses the vocabulary specified by the ontology to generate a request. Then, it starts looking for a matching service. Once discovered, the client invokes the service and retrieves the solution resource from the provider. Figure 4.4 shows the activities of the client.

![Client activity diagram](image)

**Fig. 4.4.** Client activity diagram

4.5.2 The provider

The provider integrates a DPWS device. It may host one or more services. To simplify our scenario, we suppose that a provider hosts just one service. Its role is to offer the solution service. Before creating the desired service, the provider has to search for the ontology service provider. Once found, it generates the service name taking into consideration the vocabulary specified by the ontology. Then, the provider announces the new service. After that, it waits for probing messages. If the probe matches the service, the discovery message exchange is completed with the client. The service is invoked and the solution is delivered to the client. Figure 4.5 shows the activities of the provider.
4.5.3 The ontology service provider

The ontology service provider is a special provider. Its task is to provide users with the ontology describing the service context of the corresponding environment. Once invoked, its output is an attachment file representing the ontology in OWL. In our motivation scenario, this component will be provided by the university. A simple instance of this ontology service provider is sufficient. Complex cases with multiple instances of this provider could be studied.

4.5.4 Interactions

Figure 4.6 illustrates the interactions between a client, a provider and the ontology provider. The context ontology provider starts by sending a Hello message announcing its service. When a provider joins the network, it starts searching for the CO service by sending a probe message. Once found, the provider invokes it from the CO provider. Using the retrieved ontology, the provider generates the desired service and sends a Hello message for it. When a client joins the network, it proceeds with the same steps as the provider until retrieving the context ontology. Then, the client generates the service request and sends a Probe message for it. If the provider matches the request, then the client will invoke the service and get the corresponding output. Appendix B lists the DPWS messages exchanged during the discovery process.
Fig. 4.6. Client/provider interactions sequence diagram
4.6 Prototype architecture

Figure 4.7 represents the components of the solution prototype.

![Prototype architecture diagram]

The prototype architecture includes three different layers:

- **DPWS protocol stack**: This layer integrates the WS-* defined in DPWS.
- **Application layer**: This layer is composed of the following modules:
  - Context ontology client module: It is a special client that searches and retrieves the context ontology from the CO provider.
  - Client module: It is instantiated when a user starts the application with a client profile. It is associated with a DPWS client.
  - Provider module: It is instantiated when a user starts the application with a provider profile. It is associated with a DPWS device. The client/provider module interacts with the CO client module in order to retrieve the CO.
4.7 Summary

- Graphical user interface (GUI) module: It helps the user select the vocabulary for the request or service specification.
- Semantic layer: This layer includes the ontology manager module. It handles the OWL ontology. Its tasks are to navigate through the ontology, extract the instances and deliver them to the user for request/service generation.

We implemented a Java-based prototype without the GUI module. We used a set of libraries for this purpose:

- JMEDS (Java Multi Edition DPWS Stack): It is a Java implementation of the DPWS protocol stack.
- Apache Jena: The Jena framework includes an ontology API for handling OWL and RDFS ontologies. It is used for manipulating the context ontology.

4.7 Summary

In this chapter, we presented a new design for a distributed architecture for context-aware web services starting from the motivation of a campus scenario in which students or professors trying to exchange a solution about a specific problem. The analysis of the scenario highlighted a set of requirements. First, the architecture is distributed and integrates mobile and heterogeneous devices. Then, the services deployed are domain-related. To fulfill these characteristics, we proposed an ontology-based context model for describing campus related services. This model can be extrapolated to other domains by instantiating the ontology concepts with the corresponding environment. This model was integrated within the DPWS discovery mechanism in order to allow users to generate services or requests from the vocabulary defined by the ontology. Three components participate in the discovery mechanism: the client, the provider and the context ontology provider. We specified the activities as well as the interactions of these elements. Finally we illustrated an application prototype.

Validation

The validation of the distributed architecture for context-aware web services is presented in this chapter. The objective is to prove that using the context ontology results in better performance on the service discovery mechanism. The architecture is validated via discrete-event simulation. The evaluated metrics are the discovery ratio, the mean discovered services per request, and the overhead of probes for the context ontology service.

5.1 Introduction

The novelty of our solution for service discovery is the use of ontology to share the vocabulary necessary for the description of the services and the requests. The objective of the validation is to demonstrate that the use of the ontology produces better performance in terms of service discovery. To accomplish our aim, first a validation methodology should be adopted. It consists in comparing scenarios with the context ontology (CO) service and others where users introduce freely their petitions. Then discrete-event simulations will be performed. Hence, various user behaviors have to be simulated. They include user session duration, user inter-arrival time, client inter-request time and service popularity. In order to choose the most adequate models for our case of study, various behavior models will be discussed. The simulations aim to evaluate a set of metrics related to the discovery process: discovery ratio and number of discovered services per request. The overhead generated by the use of the context ontology service will be also measured.

First, previous tasks that were performed and discarded are described in section 5.2. In section 5.3, the validation methodology for evaluating the use of the context ontology is detailed. Section 5.4 defines the discrete-event simulations settings. Simulation conditions, metrics, parameters and user behavior models are specified. In section 5.5 simulations results are discussed. Finally, we include the summary of the chapter.
5.2 Previous tasks

Before aiming our principle objective, we performed other tasks that were relevant for the definition of our goal as well as the validation methodology. First, we designed a modified architecture in which services are grouped according to specific criterion. The grouping process includes the selection of a group leader and therefore a group management strategy. The tested hypothesis was that grouping services offers better performance during the discovery process than architecture with ungrouped services. After comparing the two designs, we concluded not only that the hypothesis is false but also that the grouping process generates an overhead. The second task we performed was building an emulator based on the prototype described in the previous chapter. We deployed a wireless network with three machines: the provider generator, the client generator and the context ontology service provider. Performing various experiments ended with the crash of the provider generator. The problem was that the library used for the DPWS protocol stack does not support running a high number of provider instances on the same machine. Therefore the emulator development was canceled.

5.3 Validation methodology

Our objective is to demonstrate that using the context ontology enhances the number of matched requests. In order to fulfill this goal, we will analyze the discovery mechanism and the generation process by comparing from one hand the use of the ontology as source of vocabulary and from the other hand users specifying freely the keywords that define their petitions. This comparison will be accomplished through discrete-event simulations that integrate the behaviors and the interactions of the components described in section 4.5. The behavior of a user defining freely his service or request is simulated as follows: First, the user selects keywords from the ontology instances. Then, the service is generated from random synonyms of the selected keywords. These synonyms are obtained via Wordnet [64, 76]. Wordnet is a lexical database for the English language; it groups words in sets of synonyms.

5.4 Validation via discrete-event simulation

Since we discarded the validation via an emulation as explained in section 5.2, we will validate our solution via a discrete-event simulation. In this section, the simulation settings are defined. First, general conditions are set. Then, the validation metrics are specified. After that, the simulation parameters are described. User behavior models are then discussed in order to select the most adequate ones for our case. Finally an overview of the simulator is presented.
5.4 Simulation conditions

The simulations are subject to the following conditions:

- The network traffic is not simulated.
- Resolve and Resolve Match messages are not simulated because they are not mandatory for the discovery process.
- In an ontology source simulation, each user performs one probe for context ontology service at startup.
- A provider hosts only one service. A DPWS device is associated with a service. To simplify the simulation, we consider that probing for a service is equivalent to probing for a device. A request is considered matched if it receives a positive probe match message from a provider device.

5.4.2 Validation metrics

1. **Discovery ratio**: It is the ratio of the requests that matched at least one service to the total requests in a simulation (equation (5.1)). The discovery ratio \( \in [0 : 1] \).
   
   \[
   \text{discovery ratio} = \frac{Q_m}{Q} = \frac{\sum_{i=1}^{N} qm_i}{\sum_{i=1}^{N} q_i}
   \]

   - \( N \): Number of clients.
   - \( q_i \): Number of requests for client \( i \).
   - \( qm_i \): Number of requests that matched at least one service for client \( i \).
   - \( s_{ij} \): Number of matched services for request \( j \) of client \( i \).
   - \( Q_m = \sum_{i=1}^{N} qm_i \): Total requests that matched at least one service.
   - \( Q = \sum_{i=1}^{N} q_i \): Total requests.

2. **Mean discovered services per request**: A request can be matched with more than one service. We will evaluate the mean number of discovered services per matched requests (equation (5.2)). Since we are considering only the matched requests, the mean discovered services per request \( \geq 1 \).

   \[
   \text{mean discovered services per request} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{qm_i} s_{ij}}{\sum_{i=1}^{N} qm_i}
   \]

3. **Probe overhead**: It is the ratio of probe messages generated for the context ontology service discovery to the total probe messages (equation (5.3)).

   - \( M \): Number of providers.
   - \( K = M + N \): Number of users.
Validation

- \( p^k_{\text{ontology}} \): Number of probe messages for ontology service for user \( k \).
  In our scenario, \( p^k_{\text{ontology}} = 1 \). In fact, each user performs one probe for ontology service at startup.
- Probe messages for solution services:
  \( p_{\text{solution}} = \sum_{i=1}^{N} q_i \)
- Probe messages for ontology source scenario:
  \( p_{\text{ontology}} = \sum_{i=1}^{N} q_i + \sum_{k=1}^{K} p^k_{\text{ontology}} = \sum_{i=1}^{N} q_i + K \)

\[
\text{Probe overhead} = 100 \times \frac{p_{\text{ontology}} - p_{\text{solution}}}{p_{\text{ontology}}} = 100 \times \frac{K}{\sum_{i=1}^{N} q_i + K}
\]  

5.4.3 Simulation parameters

- **Simulation time**: It is fixed to a working day length (8 hours).
- **Service vocabulary source**: Varying this parameter will show the advantage of using the ontology as a source for generating users’ petitions. As stated before, the source can be the context ontology or synonyms from the Wordnet lexical database. Varying the number of synonyms that a word may have controls the degree of freedom with which users introduce keywords for specifying their petitions. Since this number varies depending on the word, we fixed the range of the synonyms set. In our case, the number of synonyms varies between 1 and 6 and has a mean of 2.5. Hence, we will consider (1, 2 and 5) as a range for the number of synonyms. The case where there is no limitation for the number of synonyms is also taken into consideration (no_max).
- **Number of service types**: This parameter controls the heterogeneity of the services provided and requested in a simulation scenario. Service type is the service name generated by the concatenation of the selected keywords. A user will select randomly a service type when defining a request or a service. The set of types from which a user can select depends on the ontology instances. The range of this set will be fixed (1, 5, 10 and 20). We also run simulations with all the possible combination of service names (**all**). In our case the **all** value is 60 service types.
- **User Mix**: This parameter controls the distribution of users’ profiles. We define three mixes: providing (75% of providers and 25% of clients), consuming (25% of providers and 75% of clients) and equal (50% of provider and 50% of users).

Table 5.1 summarizes the simulation parameters with their corresponding ranges.
5.4 Validation via discrete-event simulation

Table 5.1. Simulation parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation time (t_{sim})</td>
<td>8 hours (working day)</td>
</tr>
<tr>
<td>Service vocabulary source</td>
<td>Ontology source</td>
</tr>
<tr>
<td>(vocab_{src})</td>
<td>Wordnet synonyms ({1, 2, 5, \text{no_max}})</td>
</tr>
<tr>
<td>Number of service types</td>
<td>({1, 5, 10, 20, \text{all}})</td>
</tr>
<tr>
<td>(stype)</td>
<td>Providing mix ((75% \text{ providers}, 25% \text{ clients}))</td>
</tr>
<tr>
<td></td>
<td>Consuming mix ((25% \text{ providers}, 75% \text{ clients}))</td>
</tr>
<tr>
<td>User mix (mix)</td>
<td>Equal mix ((50% \text{ providers}, 50% \text{ clients}))</td>
</tr>
</tbody>
</table>

5.4.4 User behavior models

In order to simulate our case of study, we have to specify some parameters of user behavior. These parameters are the user arrival rate, the user session duration, the client think-time (or inter-request time) and the service popularity. Various works related to user behavior characterization have been studied in order to select the most adequate statistical distributions for our case.

Balachandran et al. [5] analyzed user behavior in a public-area wireless network using workload captured at the ACM SIGCOMM’01 conference. User arrival is modeled as a Two-State Markov-Modulated Poisson Process (MMPP2) with ON and OFF states. The OFF state is when there are no arrivals into the system. The mean inter-arrival rate during the ON state is 38 seconds. The mean duration of the OFF state is 6 minutes. The authors also concluded that the user session duration follows a general Pareto distribution with a shape parameter of 0.78 and a scale parameter of 30.76. This behavior is typical of settings such as classrooms, meeting, etc. Therefore, it can be extrapolated to our case of study.

Casalicchio [17] modeled web workload. The user thinking time (inter-request time) follows a Pareto distribution with a shape parameter of 1.14 and a scale parameter of 2. The behavior of the clients in our case can be assimilated to web users.

Gill et al. [34] analyzed traffic of a video sharing service (YouTube) using data collected at the University of Calgary campus network. The study shows that video references at the campus fit a Zipf-like distribution with a skew parameter of 0.56. Since the data are collected from a specific population of users, the academic community, video popularity can be viewed as service popularity within the context of the campus network.

Duarte et al. [31] characterized access patterns in blogospace. The study shows that the popularity of objects follows a general power law with skew parameter 0.97. The inter-arrival time is fitted with a Weibull distribution.
Benevenuto et al. [9] analyzed user behavior in social networks. User session modeling shows that the inter-arrival time and the inter-request time fit a lognormal distribution. The number of requests per session follows a Zipf distribution.

The methodology of analyzing and selecting the adequate distributions is based on the work done by Guerrero et al. [39]. Table 5.2 summarizes the discussed user behavior models.

**Table 5.2. Summary of discussed user behavior models**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-arrival time</td>
<td>MMPP2 lognormal Weibull - -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session time</td>
<td>Pareto - - - -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter-request time</td>
<td>- lognormal - Pareto -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requests per session</td>
<td>- Zipf-like - Inverse Gaussian -</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popularity</td>
<td>- - Power law - Zipf-like</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To sum up, we are considering an MMPP2 distribution for the inter-arrival time (see appendix C for a detailed explanation of the MMPP2 distribution), a Pareto for the session time and the inter-request time and a Zipf-like for the service popularity. Table 5.3 shows the selected distributions for the simulations.

**Table 5.3. Selected statistical distributions for the characterization of user behavior**

<table>
<thead>
<tr>
<th>Simulated behavior</th>
<th>Distribution</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>User inter-arrival time</td>
<td>$t_{inter_arrival} \sim MMPP2$ [5]</td>
<td>mean inter-arrival ON state: $\mu_{on} = 38s$ mean duration OFF state: $\mu_{off} = 6\text{mn}$</td>
</tr>
<tr>
<td>User session duration</td>
<td>$d_{session} \sim Par$ [5]</td>
<td>shape: $\alpha = 0.78$ scale: $\beta = 30.78$</td>
</tr>
<tr>
<td>Client inter-request time</td>
<td>$t_{inter_req} \sim Par$ [17]</td>
<td>shape: $\alpha = 1.4$ scale: $\beta = 2$</td>
</tr>
<tr>
<td>Service type popularity</td>
<td>$service_pop \sim Z$ [34]</td>
<td>skew: $s = 0.56$</td>
</tr>
</tbody>
</table>
5.4.5 Simulator implementation

We implemented a Java-based simulator for our architecture with a set of referenced libraries:

- SSim (A Simple Discrete-Event Simulation Library) \(^1\): it is a utility library that implements a simple discrete-event simulator.
- Colt \(^2\): It is a set of Open Source Libraries for High Performance Scientific and Technical Computing in Java. It is used for generating the statistical distributions.
- Java Modelling Tools (JMT) \(^3\): It is a framework for performance evaluation, system tuning, capacity planning and workload characterization studies. It is used for generating the MMPP2 distribution.
- JWI (the MIT Java Wordnet Interface) \(^4\): It is a Java library for interfacing with Wordnet.
- Apache Jena\(^5\): The Jena framework includes an ontology API for handling OWL and RDFS ontologies.

5.5 Simulation results and discussion

In this section, we will discuss the measures for the three validation metrics described in 5.4.2.

5.5.1 Simulation identifications

Taking into account the different parameters, we will run 75 simulations. In order to identify the results, we will adopt the following nomenclature:

\[
P(mix, stype, vocab\_src) \text{ where } \begin{cases} 
mix & \in \{c, e, p\} 
stype & \in \{1, 5, 10, 20, all\} 
vocab\_src & \in \{o, w1, w2, w5, wno\} 
\end{cases}
\]

For example: \(P(c, 5, o) \Rightarrow \begin{cases} 
mix = consuming 
stype = 5 
vocab\_src = ontology 
\end{cases}\)

When \(\ast\) is used in \(P()\), it means that the measures cover all the values in the corresponding range.

\(^1\) http://www.inf.usi.ch/carzaniga/ssim/, Last accessed: June, 2013
\(^2\) http://acs.lbl.gov/software/colt/, Last accessed: June, 2013
\(^3\) http://jmt.sourceforge.net/, Last accessed: June, 2013
5.5.2 Discovery ratio

Simulations $P(c, 5, *)$

Figure 5.1 represents the discovery ratio for consuming mix, 10 service types and for different vocabulary sources. The discovery ratio decreases with the number of maximum synonyms used for the Wordnet simulations. The discovery ratio for the ontology source is equivalent to the scenario when using one synonym ($w_1$). By looking at the measures in table 5.4, we notice that the ratio for $w_1$ is a bit higher than the ontology scenario. This is explained by the fact that during the ontology scenario, there is a probing phase for the context ontology service. Providers publish their services after this phase. Therefore, requests that probe provider during this phase are not matched with the corresponding service. The scenario $w_1$ is perfect and unreal case. In fact, users cannot express their petitions with the same keywords unless previous knowledge. The scenario with the context ontology tends to the perfect scenario of $w_1$. Hence, using the ontology to share the service vocabulary enhances service discovery.

![Diagram](image-url)
Simulations $P(c,*,*)$

Figure 5.2 illustrates the variation of the discovery ratio with the number of service types for the consuming mix. For each group of measures for a particular number of service types, we notice the same behavior as in $P(c, 10, *)$. The discovery ratio decreases with $stype$. This is due to the increasing of the heterogeneity degree of the services being provided and requested within the campus environment. In some cases there is a very small difference between $w_{no}$ and $w_5$. This difference is not significant and do not highlight any special behavior. In fact, the two cases are equivalent if the words chosen in a $w_{no}$ scenario have at most five synonyms. Besides, when a word with more than five synonyms is selected, it can have at most six synonyms. Hence, the heterogeneity in service names is not affected at a large scale. Therefore, these small differences are explained by the random effect of the simulations.

Simulations $P(*,*,*)$

Figure 5.3 represents the discovery ratio covering all the corresponding ranges for the different parameters. The figure illustrates the effect of user mix on the metric. For each mix, we notice the same behavior as described in $P(c, *, *)$. Considering the measures in table 5.4, we observe that the user mix has a
small effect on the discovery ratio. The comparison between the three mixes shows that the providing mix gives the best results, then the equal and finally the consuming. This classification is valid for most of the cases. The mix effect is due to the ratio request to service. In the providing mix, the scenario generates fewer requests, but with a higher probability to be matched because of the number of services. In the consuming mix, the discovery ratio is the lowest due to the fact that there is a higher number of requests with few services. The equal mix offers an equilibrated ratio requests to services which increases the number of matched requests compared to the consuming mix. The simulation measures of the discovery ratio are presented in table 5.4.

5.5.3 Mean discovered services per request

Simulations $P(c,10,*)$

Figure 5.4 represents the mean discovered services per request for consuming mix, 10 service types and for different vocabulary sources.

Simulations with ontology source and $w_1$ show higher measures. Comparing the simulations with $w_2$, $w_5$ and $w_{no}$, we observe that there is a small increasing of the metric with the number of synonyms. Considering the discovery ratio for these last three cases, we notice that the ratio decreases. Besides, we are counting only the matched requests. Hence, a smaller number
Table 5.4. Discovery ratio measures

<table>
<thead>
<tr>
<th>mix</th>
<th>stype</th>
<th>vocab_src</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>o</td>
</tr>
<tr>
<td>equal</td>
<td>1</td>
<td>0.9830</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.7312</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.5399</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.3564</td>
</tr>
<tr>
<td></td>
<td>all(60)</td>
<td>0.1991</td>
</tr>
<tr>
<td>providing</td>
<td>1</td>
<td>0.9670</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.7670</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.5470</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.3260</td>
</tr>
<tr>
<td></td>
<td>all(60)</td>
<td>0.1760</td>
</tr>
<tr>
<td>consuming</td>
<td>1</td>
<td>0.9874</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.6568</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.4930</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.3199</td>
</tr>
<tr>
<td></td>
<td>all(60)</td>
<td>0.1557</td>
</tr>
</tbody>
</table>

Simulations P(c,*,*)

Figure 5.5 illustrates the variation of the mean discovered services per request with the number of service types for the consuming mix. For each group of measures for a particular number of service types, we notice the same behavior as in \( P(c, 10, \*) \). The mean number of discovered services per request decreases with \( stype \). This is due to the increasing of the heterogeneity of the services being provided and requested. With high values of service type, the number of discovered services tends to the minimum value (1).

Simulations P(*,*,*)

Figure 5.6 represents the mean discovered services per request covering all the corresponding ranges for the different parameters. The figure illustrates the
Fig. 5.4. Mean discovered services per request for $P(c, 10, *)$

Fig. 5.5. Mean discovered services per request for $P(c, *, *)$
Fig. 5.6. Mean discovered services per request for $P(\ast, \ast, \ast)$

effect of user mix on the metric. For each mix, we notice the same behavior as described in $P(c, \ast, \ast)$. The effect of the mix on the number of discovered services per request is not relevant. The small advantage of the providing over the equal and the consuming is explained by the rate of services to requests. The simulation measures of the mean discovered service per request are presented in table 5.5

5.5.4 Probe messages overhead

The probe messages overhead depends on the number of users and the number of search requests (equation (5.3)). Therefore, the user mix is the only parameter that affects this metric. Figure 5.7 illustrated the overhead of the probe messages for the three mixes.

Simulations with consuming mix generate the lowest overhead (8.75%). This is due to the highest number of search requests. With the equal mix scenarios, we observe an 18.21% of probe overhead, since this mix generates fewer requests than the consuming one. We measure the highest overhead (32.22%) with the providing mix because of the smaller number of search requests compared to the other two mixes.
### Table 5.5. Mean discovered services per request measures

<table>
<thead>
<tr>
<th>mix</th>
<th>stype</th>
<th>vocab_src</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>o</td>
<td>$w_1$</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td>$w_{no}$</td>
<td></td>
</tr>
<tr>
<td>equal</td>
<td>1</td>
<td>7.7639</td>
<td>7.7800</td>
<td>1.6161</td>
<td>1.1739</td>
<td>1.2309</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.3827</td>
<td>2.3752</td>
<td>1.1705</td>
<td>1.0214</td>
<td>1.0057</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.7808</td>
<td>1.7750</td>
<td>1.0548</td>
<td>1.0172</td>
<td>1.0190</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1.5212</td>
<td>1.5244</td>
<td>1.0779</td>
<td>1.0000</td>
<td>1.0588</td>
<td></td>
</tr>
<tr>
<td></td>
<td>all(60)</td>
<td>1.1313</td>
<td>1.1302</td>
<td>1.0723</td>
<td>1.0000</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>providing</td>
<td>1</td>
<td>8.3216</td>
<td>8.3643</td>
<td>1.6358</td>
<td>1.1678</td>
<td>1.1391</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.2347</td>
<td>2.2442</td>
<td>1.2189</td>
<td>1.0313</td>
<td>1.0349</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.7495</td>
<td>1.7411</td>
<td>1.0504</td>
<td>1.0465</td>
<td>1.0185</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1.5368</td>
<td>1.5350</td>
<td>1.0182</td>
<td>1.0000</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>all(60)</td>
<td>1.2955</td>
<td>1.2931</td>
<td>1.0698</td>
<td>1.0526</td>
<td>1.0000</td>
<td></td>
</tr>
<tr>
<td>consuming</td>
<td>1</td>
<td>6.7078</td>
<td>6.7175</td>
<td>1.4717</td>
<td>1.1461</td>
<td>1.1169</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.4996</td>
<td>2.4973</td>
<td>1.1714</td>
<td>1.0675</td>
<td>1.0187</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.7500</td>
<td>1.7513</td>
<td>1.0884</td>
<td>1.1152</td>
<td>1.2346</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>1.5539</td>
<td>1.5583</td>
<td>1.2321</td>
<td>1.0305</td>
<td>1.1872</td>
<td></td>
</tr>
<tr>
<td></td>
<td>all(60)</td>
<td>1.3280</td>
<td>1.3276</td>
<td>1.0231</td>
<td>1.0000</td>
<td>1.0351</td>
<td></td>
</tr>
</tbody>
</table>

#### 5.5.5 Results summary

Taking into account the different simulations performed above, we highlight the following observations:

- Sharing the context ontology allows the best matching of users requests. In fact, when using the ontology as a vocabulary source, more requests are matched (discovery ratio) with more services (mean discovered services per request).
- Increasing the number of synonyms in Wordnet scenarios gives more freedom to users when specifying their petitions. Therefore, the number of matched requests decreases as well as the number of discovered services.
- The heterogeneity of the services affects the discovery performance. Hence, the more service types are provided or requested, the lower the discovery ratio and the discovered services per request are.
- The user mix does not have a notable effect either on the discovery ratio or the mean discovered services per request.
• The probe overhead generated when using the context ontology depends on the user mix. The consuming mix produces the lowest overhead while the providing one produces the highest.

Other simulations were performed by varying the statistical distributions as well as their parameters. For example we run simulations with an exponential user arrival and session duration and with uniform service popularity. In all these simulations, we noticed similar behaviors as the ones discussed with the selected distributions.

5.6 Summary

This chapter exposed the validation of our distributed architecture for context-aware web services. We demonstrated that sharing the vocabulary for service and request description via ontology enhances the discovery process. This was accomplished by comparing scenarios with the CO service and others where users specify freely their petitions. The latter were simulated by selecting synonyms from the Wordnet lexical database. A set of metrics was evaluated via discrete-event simulations. The discovery ratio measures the ratio of the matched requests to the total number of requests. The mean number of discovered services per request represents the quantity of services matched per request. An overhead metric was also introduced. It consists of the ratio
of probes for the context ontology service to the total probe messages. The simulations were driven by a set of user behavior models. A study of existing models was necessary to select the most adequate ones for our case. An MMPP2 distribution was selected for the user inter-arrival time, a Pareto distribution for the user session duration and the client inter-request time, and a Zipf-like distribution for the service popularity. A set of parameters was defined to control the simulation: simulation time, the number of service type, the number of synonyms in Wordnet scenario and the user mix. The evaluation showed that scenarios with the context ontology service produced better discovery ratio and matched requests with more services than the ones with Wordnet (except for the $w_1$). We also concluded that increasing the service types affects negatively the discovery ratio as well as the number of discovered services. However, the user mix effect is only notable when measuring the probe overhead. In fact with less search requests, higher overhead is generated. With the validation phase, we proved our hypothesis by confirming that sharing the vocabulary for service description via ontology, the discovery mechanism is enhanced.
Part III

Conclusions
Conclusion and open problems

In this chapter, we summarize the contributions of this thesis and discuss future work.

6.1 Thesis summary

This thesis demonstrated that sharing the vocabulary used for service description enhances context-aware service discovery. This was fulfilled by designing a distributed architecture for enhanced context-aware web services. The starting point is a motivation scenario that describes university students trying to exchange a solution about a specific problem within the campus environment. To solve our scenario, two studies were performed. First, in chapter 2, we surveyed context-aware systems. Core concepts such as context and context-awareness were defined according to Dey’s vision. A comparison of context modeling techniques has shown that the ontology-based one has a higher and more formal expressiveness. We also presented an abstract layered architecture that gives an overview of the capabilities of such systems. Existing approaches were discussed according to criteria such as context model, architecture, and application domain. Then, we researched service discovery mechanisms in chapter 3. We covered different categories of services: general services, web services, and semantic web services. For the first one, we reviewed Jini, the Service Location Protocol (SLP), Universal Plug and Play (UPnP), and the Bluetooth service discovery protocol. In the web service category, we described Universal Description Discovery and Integration (UDDI) and Devices Profile for Web Services (DPWS). OWL-S and WSMO were presented as discovery approaches for semantic web services. We also discussed research approaches that enhanced discovery mechanisms with the context-awareness feature. Based on the previous two surveys, we designed our solution in chapter 4. The analysis of the motivation scenario allowed us first to propose an ontology-based model for the service vocabulary and then to specify the architecture components and their behaviors. DPWS approach
was chosen as a framework for providing and requesting web services. In fact, DPWS is adapted for resource-constraint devices. It also has a distributed discovery mechanism based on WS-discovery. Then, we integrated the context model, the users’ behaviors and the discovery mechanism to obtain a prototype of the proposed solution. To validate our hypothesis, we performed a series of discrete-event simulations depicted at chapter 5. The validation methodology consisted in comparing scenarios using the context ontology as vocabulary source and others that use synonyms from Wordnet. To evaluate our approach, we defined three metrics. The first one is the discovery ratio. It calculates the ratio of requests that matched at least one service to the total requests. The second one is the mean number of discovered services per request. We also evaluated the overhead generated by the probe messages for the context ontology service. Parameters were defined in order to control the simulation: simulation time, number of synonyms, number of service types, and user mix. After studying different user behavior models, we selected the most adequate statistical distributions for our simulations. Hence, we used an MMPP2 distribution for the user inter-arrival time, a Pareto distribution for the user session duration and the client inter-request time, and a Zipf-like distribution for the service popularity. The simulation results have shown that scenarios with the context ontology service produced better discovery ratio and matched requests with more services than the ones with Wordnet. We also concluded that increasing the service types affects negatively the discovery ratio as well as the number of discovered services. However, the user mix effect is only notable when measuring the probe overhead. In fact with less search requests, higher overhead is generated.

6.2 Contributions

The main contributions of this thesis are:

- A motivation scenario for active user behavior. We described a scenario where users acquire the ability of providing their own services via their devices. This scenario could be an appropriate new market scenario for real context-aware applications based on current Smartphone capabilities.
- A survey on context-aware systems. We covered different aspects of context-aware systems: core definitions, modeling techniques, and architecture capabilities. We also discussed various existing systems.
- A survey on service discovery mechanisms. We reviewed industry and consortia supported mechanisms as well as research approaches. This study analyzed the context-aware feature in service discovery approaches.
- A new ontology-based context model for service vocabulary. This model was designing after analyzing the requirements of the motivation scenario. The used technique was chosen based on a comparison of various modeling approaches.
• A new design of distributed architecture for context-aware web services. This architecture constitutes the solution for our case of study. It integrates an ontology based model for service description and a distributed discovery mechanism based on DWPS.
• An architecture prototype. We illustrated a prototype for a user application that includes a client and a provider profile.
• A set of performance metrics. We defined discovery-related metrics such as the discovery ratio and the mean number of discovered services per request. The overhead generated by the probe messages was also measured.
• A shared vocabulary validation. Due to a discrete-event simulation results, we proved our hypothesis confirming that sharing vocabulary for service description enhances service discovery.

6.3 Future work and open problems

There are some open problems in our work that can be further researched:

• The implementation of an application prototype. Implementing a mobile application based on the proposed prototype is the first step towards the deployment of the architecture.
• The architecture deployment and test in a real campus environment. These tasks can be accomplished at different scales. First, we will test the functionalities of the architecture on a small group of users. Then, we will evaluate the scalability of the prototype.
• The characterization of user behavior. The user behavior models used for our simulations were selected as an approximation for our scenario. Hence, we need to monitor the utilization of the architecture in a real environment in order to have better fitted statistical distributions.
• The user feedback evaluation. Evaluating the proposed solution can also be performed via user feedbacks. This will help to improve the prototype.
• Network performance evaluation. Since network traffic was not simulated in the validation phase and we noticed that our solution generates a probe overhead, we have to study the network performance.
• The port of the architecture to another environment. The starting point was a campus case. Hence, we have to demonstrate that the context model as well as the architecture is valid for any kind of environment in which the users have similar behaviors.
Context ontology

In this appendix, the context ontology used for service and request description is listed in Web Ontology Language (OWL) (listing A.1)

Listing A.1. Context ontology

```xml
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns="http://acsic.uib.es/ContextService.owl#"
  xmlns:swrl="http://www.w3.org/2003/11/swrl#"
  xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
  xmlns:dfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://acsic.uib.es/ContextService.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="User">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      root</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="Petition"/>
  <owl:Class rdf:ID="Resource"/>
  <owl:Class rdf:ID="Category"/>
  <owl:Class rdf:ID="Type"/>
  <owl:Class rdf:ID="Role"/>
  <owl:Class rdf:ID="Field"/>
  <owl:ObjectProperty rdf:ID="hasRole">
    <rdfs:domain rdf:resource="#Role"/>
    <rdfs:range rdf:resource="#Role"/>
    <dfs:domain rdf:resource="#User"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="express">
    <dfs:domain rdf:resource="#User"/>
    <dfs:range rdf:resource="#Petition"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasCategory">
    <dfs:domain rdf:resource="#Category"/>
    <dfs:range rdf:resource="#Category"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasResources">
    <dfs:domain rdf:resource="#Resource"/>
    <dfs:range rdf:resource="#Resource"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasField">
    <dfs:domain rdf:resource="#Category"/>
    <dfs:range rdf:resource="#Category"/>
  </owl:ObjectProperty>
</rdf:RDF>
```
A Context ontology

```xml
<owl:ObjectProperty rdf:ID="hasType"/>
<owl:ObjectProperty rdf:ID="hasField"/>
<owl:ObjectProperty rdf:ID="hasRole"/>
<User rdf:ID="Professor"/>
<hasRole>
<Role rdf:ID="Requester"/>
</hasRole>
<hasRole>
<Role rdf:ID="Provider"/>
</hasRole>
<express>
<Petition rdf:ID="Solution"/>
</express>
</User>
```
<rdf:RDF xmlns:hasField="http://example.com/hasField#" xmlns:hasType="http://example.com/hasType#" xmlns:hasCategory="http://example.com/hasCategory#" xmlns:hasResources="http://example.com/hasResources#">
  <Category>
    <hasField rdf:resource="#Chemistry"/>
    <hasType rdf:resource="#Project"/>
    <hasType rdf:resource="#Publication"/>
    <hasField rdf:resource="#Computer"/>
    <hasField rdf:resource="#Physics"/>
    <hasType rdf:resource="#Analysis"/>
    <hasField rdf:resource="#Biology"/>
    <hasType rdf:resource="#Report"/>
    <hasType rdf:resource="#Subscription"/>
  </Category>
  <hasCategory>
    <hasResources>
      <Resource rdf:ID="Link"/>
    </hasResources>
    <hasResources>
      <Resource rdf:ID="Meeting"/>
    </hasResources>
    <hasResources>
      <Resource rdf:ID="Book"/>
    </hasResources>
    <hasResources>
      <Resource rdf:ID="File"/>
    </hasResources>
  </hasCategory>
  <express>
    <Petition rdf:ID="Problem">
      <hasResources rdf:resource="#Link"/>
      <hasResources rdf:resource="#Meeting"/>
      <hasResources rdf:resource="#Book"/>
      <hasResources rdf:resource="#File"/>
      <hasCategory rdf:resource="#Course"/>
      <hasCategory rdf:resource="#Administration"/>
    </Petition>
  </express>
  <User rdf:ID="Staff">
    <express rdf:resource="#Solution"/>
    <express rdf:resource="#Problem"/>
    <hasRole rdf:resource="#Requester"/>
    <hasRole rdf:resource="#Provider"/>
  </User>
  <User rdf:ID="Student">
    <hasRole rdf:resource="#Requester"/>
    <hasRole rdf:resource="#Provider"/>
    <express rdf:resource="#Solution"/>
    <express rdf:resource="#Problem"/>
  </User>
  <Type rdf:ID="Demonstration"/>
</rdf:RDF>

<-- Created with Protege (with OWL Plugin 3.4.6, Build 613)
http://protege.stanford.edu -->
In this appendix, Web service descriptions are listed in B.1. Messages exchanged during discovery and invoking phases are described in B.2.

B.1 Web services descriptions

Listing B.1. Context ontology service WSDL document

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- This is an auto-generated WSDL from JMEDS. 
Copyright (c) 2009 MATERIA Information & Communications and TU Dortmund, Dept. of 
Computer Science, Chair 4, Distributed Systems. 
All rights reserved. 
JMEDS and the accompanying materials are made available under the terms of the 
Eclipse Public License v1.0 
which accompanied this distribution, and is available at 
  <wsdl:types>
    <xs:schema attributeFormDefault= "unqualified" elementFormDefault="qualified" targetNamespace="http://acsic.uib.es/onto">
      <xs:element name= "OutputAttachment" type="tns:AttachmentType"/>
      <xs:complexType name= "AttachmentType">
        <xs:sequence>
          <xs:element name="File" type="xs:base64Binary"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name= "GetAttachmentOperationMessage"/>
  <wsdl:message name= "GetAttachmentOperationResponseMessage"/>
  <wsdl:part element="tns:GetAttachmentOperation" name="parameters"/>
  <wsdl:portType name="ContextOntologyService">
    <wsdl:operation name="GetAttachmentOperation">
      ContextOntologyService/GetAttachmentOperation"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```
Listing B.2. Example of a solution service WSDL document
B.2 DPWS messages

B.2.1 Hello

Listing B.3. Hello message

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<s12:Envelope xmlns:dpws="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsd="http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01">
  <s12:Header>
    <wsa:Action>
      http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Hello
    </wsa:Action>
    <wsa:MessageID>
      urn:uuid:64095760-8bed-11e2-bf7f-280936d322de
    </wsa:MessageID>
    <wsa:To>
    </wsa:To>
    <wsd:AppSequence InstanceId="1363186247" MessageNumber="1"/>
  </s12:Header>
  <s12:Body ID="BID1">
    <wsd:Hello>
      <wsd:Types xmlns:i1="http://acsic.uib.es/solutionservice"
          xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof">dpws:Device
          wsdp:Device i1:StudentDevice </wsd:Types>
    </wsd:Hello>
  </s12:Body>
</s12:Envelope>
```
B.2.2 Bye

Listing B.4. Bye message

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<s12:Envelope xmlns:dpws= "http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa= "http://www.w3.org/2005/08/addressing"
xmlns:wsd= "http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01">
 <s12:Header>
  <wsa:MessageID>urn:uuid:7651ab70-8bed-11e2-bf86-280936d322de</wsa:MessageID>
  <wsa:To>urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01</wsa:To>
  <wsd:AppSequence InstanceId= "1363186247" MessageNumber="4"/>
 </s12:Header>
 <s12:Body ID="BID1">
  <wsd:Bye>
   <wsa:EndpointReference>
    <wsa:Address>urn:uuid:63fc8620-8bed-11e2-bf7e-280936d322de</wsa:Address>
   </wsa:EndpointReference>
   <wsd:XAddrs/>
  </wsd:Bye>
 </s12:Body>
</s12:Envelope>
```

B.2.3 Probe and probe match

Matching types and scopes

This part is a transcription of the matching types and scopes specification in WS-discovery [71].
```
... A Probe includes zero, one, or two constraints on matching Target Services: a set of Types and/or a set of Scopes. A Probe Match MUST include a Target Service if and only if all of the Types and all of the Scopes in the Probe match the Target Service.

A Type T1 in a Probe matches Type T2 of a Target Service if the QNames match. Specifically, T1 matches T2 if all of the following are true:

- The namespace [Namespaces in XML 1.1] of T1 and T2 are the same.
- The local name of T1 and T2 are the same.

(The namespace prefix of T1 and T2 is relevant only to the extent that it identifies the namespace.)

A Scope S1 in a Probe matches Scope S2 of a Target Service per the rule indicated within the Probe. This specification defines the following matching rules. Other matching rules MAY be used, but if a matching rule is not recognized by a receiver of the Probe, S1 does not match S2 regardless of the value of S1 and/or S2.

http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/rfc3986
```
Using a case-insensitive comparison,

- The scheme [RFC 3986] of S1 and S2 is the same and
- The authority of S1 and S2 is the same and
Using a case-sensitive comparison,
- The path segments of S1 is a segment-wise (not string) prefix of the path segments of S2 and
- Neither S1 nor S2 contain the "." segment or the ".." segment.
All other components (e.g., query and fragment) are explicitly excluded from comparison. S1 and S2 MUST be canonicalized (e.g., unescaping escaped characters) and trailing slashes ("/") MUST be removed before using this matching rule.

Note: this matching rule does NOT test whether the string representation of S1 is a prefix of the string representation of S2. For example, "http://example.com/abc" matches "http://example.com/abc/def" using this rule but "http://example.com/a" does not.

http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/uuid
S1 and S2 are universally-unique identifier (UUID) based URN [RFC 4122] scheme URIs and each of the unsigned integer fields [RFC 4122] in S1 is equal to the corresponding field in S2, or equivalently, the 128 bits of the in-memory representation of S1 and S2 are the same 128 bit unsigned integer.

http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/ldap
Using a case-insensitive comparison, the scheme of S1 and S2 is "ldap" and the host and the port [RFC 3986] of S1 and S2 are the same and the RDNSequence [RFC 4514] of the dn [RFC 4516] of S1 is a prefix of the RDNSequence [RFC 4514] of the dn [RFC 4516] of S2.

http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/strcmp0
Using a case-sensitive comparison, the string representation of S1 and S2 is the same.

http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/none
With this rule the Probe matches the Target Service if and only if the Target Service does not have any Scopes. When a Probe specifies this rule it MUST NOT contain any Scopes. [...]"

**Probe**

**Listing B.5. Probe message**

```xml
<s12:Envelope xmlns:dpws="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsd="http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01">
  <s12:Header>
    <wsa:MessageID>urn:uuid:7deab810-d35d-11e2-bf93-5ae1dbc2c044</wsa:MessageID>
    <wsa:To>urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01</wsa:To>
  </s12:Header>
  <s12:Body>
    <wsd:Probe>
      <wsd:Types xmlns:i1="http://acsic.uib.es/solutionservice">
        i1:ComputerPracticalExcercice4</wsd:Types>
    </wsd:Probe>
</s12:Envelope>
```
Probe match

Listing B.6. Probe match message

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<s12:Envelope xmlns:dpws="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
    xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsd="http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01">
    <s12:Header>
        <wsa:MessageID>urn:uuid:6c154720-8bed-11e2-bf84-280936d322de</wsa:MessageID>
        <wsa:RelatesTo>urn:uuid:6c0c6d80-8bed-11e2-804a-faacc466482c</wsa:RelatesTo>
        <wsa:To>urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01</wsa:To>
    </s12:Header>
    <s12:Body ID="BID1">
        <wsd:ProbeMatches>
            <wsd:ProbeMatch>
                <wsa:EndpointReference>
                    <wsa:Address>urn:uuid:63fc8620-8bed-11e2-bf7e-280936d322de</wsa:Address>
                </wsa:EndpointReference>
                <wsd:Types xmlns:i172="http://acsic.uib.es/solutionservice"
                    xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof">
                    i172:StudentDevice dpws:Device wsdp:Device
                </wsd:Types>
                <wsd:XAddrs>
                    http://130.206.131.40:53119/DefaultDevice
                    http://[fe80:0:0:0:21e:c9ff:fe2d:37ae]:51205/DefaultDevice
                </wsd:XAddrs>
                <wsd:MetadataVersion>1363186250</wsd:MetadataVersion>
            </wsd:ProbeMatch>
        </wsd:ProbeMatches>
    </s12:Body>
</s12:Envelope>
```

B.2.4 Get device metadata

Listing B.7. Get device metadata message

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<s12:Envelope xmlns:dpws="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
    xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <s12:Header>
        <wsa:MessageID>urn:uuid:6cb9c480-8bed-11e2-809b-faacc466482c</wsa:MessageID>
        <wsa:To>urn:uuid:63fc8620-8bed-11e2-bf7e-280936d322de</wsa:To>
    </s12:Header>
    <s12:Body/>
</s12:Envelope>
```
Listing B.8. Get device metadata response message

```xml
<xml version="1.0" encoding="UTF-8" standalone="no"/>
<s12:Envelope xmlns:dpws= "http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
<s12:Header>
  <wsa:RelatesTo>urn:uuid:6cb9c480-8bed-11e2-809b-faacc466482c</wsa:RelatesTo>
</s12:Header>
<s12:Body>
  <wsx:Metadata>
    <wsx:MetadataSection Dialect="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/ThisModel">
      <dpws:ThisModel>
        <dpws:Manufacturer xml:lang="en-US">MATERNA GmbH</dpws:Manufacturer>
        <dpws:ManufacturerUrl>http://dpws.materna.de/</dpws:ManufacturerUrl>
        <dpws:ModelName xml:lang="en-EN">DefaultDevice</dpws:ModelName>
        <dpws:ModelNumber>1</dpws:ModelNumber>
        <dpws:ModelUrl>http://dpws.materna.de/model</dpws:ModelUrl>
        <dpws:PresentationUrl>
          http://dpws.materna.de/model/presentation.html
        </dpws:PresentationUrl>
      </dpws:ThisModel>
    </wsx:MetadataSection>
    <wsx:MetadataSection Dialect="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/ThisDevice">
      <dpws:ThisDevice>
        <dpws:FriendlyName xml:lang="en-US">StudentDevice</dpws:FriendlyName>
        <dpws:FirmwareVersion>1</dpws:FirmwareVersion>
        <dpws:SerialNumber>1</dpws:SerialNumber>
      </dpws:ThisDevice>
    </wsx:MetadataSection>
    <wsx:MetadataSection Dialect="http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/Relationship">
      <dpws:Host>
        <wsa:EndpointReference>
          <wsa:Address>urn:uuid:63fc8620-8bed-11e2-bf7e-280936d322de</wsa:Address>
        </wsa:EndpointReference>
          i173:StudentDevice dpws:Device wsdp:Device
        </dpws:Types>
      </dpws:Host>
      <dpws:Hosted>
        <wsa:EndpointReference>
        </wsa:EndpointReference>
      </dpws:Hosted>
    </wsx:MetadataSection>
  </wsx:Metadata>
</s12:Body>
</s12:Envelope>
```
B.2.5 Get service metadata

Listing B.9. Get service metadata message

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?
<s12:Envelope xmlns:dpws= "http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
  <s12:Header>
    <wsa:MessageID>urn:uuid:6cbb7230-8bed-11e2-809c-faacc466482c</wsa:MessageID>
    <wsa:To>http://130.206.131.40:44080/SolutionService</wsa:To>
  </s12:Header>
  <s12:Body>
    <wsx:GetMetadata/>
  </s12:Body>
</s12:Envelope>
```

Listing B.10. Get service metadata response message

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<s12:Envelope xmlns:dpws= "http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
  <s12:Header>
    <wsa:RelatesTo>urn:uuid:6cbb7230-8bed-11e2-809c-faacc466482c</wsa:RelatesTo>
  </s12:Header>
  <s12:Body>
    <wsx:Metadata>
      <wsx:MetadataSection Dialect= "http://schemas.xmlsoap.org/wsd1">
      </wsx:MetadataSection>
      <wsx:MetadataSection Dialect= "http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/Relationship">
        <dpws:Host>
          <wsa:EndpointReference>
            <wsa:Address>urn:uuid:63fc8620-8bed-11e2-bf7e-280936d322de</wsa:Address>
          </wsa:EndpointReference>
        </dpws:Host>
        <dpws:Hosted>
          <wsa:EndpointReference>
          </wsa:EndpointReference>
        </dpws:Hosted>
      </wsx:MetadataSection>
      <wsx:MetadataSection Dialect= "http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01/resources">
        <dpws:Device>
          <dpws:DeviceId>SolutionService</dpws:DeviceId>
        </dpws:Device>
      </wsx:MetadataSection>
    </wsx:Metadata>
</s12:Envelope>
```
B.2.6 Invoke

Listing B.11. Invoke message

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<envelope xmlns:dpws= "http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <header>
    <wsa:MessageID>urn:uuid:6ce604a0-8bed-11e2-80b3-faacc466482c</wsa:MessageID>
    <wsa:To>http://130.206.131.40:44080/SolutionService</wsa:To>
  </header>
  <body/>
</envelope>
```

Listing B.12. Invoke response message

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<envelope xmlns:dpws= "http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01"
xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <header>
    <wsa:RelatesTo>urn:uuid:6ce604a0-8bed-11e2-80b3-faacc466482c</wsa:RelatesTo>
  </header>
  <body>
    <resource xmlns:i52="http://acsic.uib.es/solutionservice">Solution</resource>
  </body>
</envelope>
```
Two-State Markov-Modulated Poisson Process (MMPP2)

In this appendix, we detail the MMPP2 distribution as explained in [84].

MMPP(2) is a continuous-time Markov chain that jumps between two states and the active state determines the current rate of service. For example, one state may be associated with slow inter-arrival times (state 0), the other may have fast inter-arrival times (state 1). As time passes, the MMPP(2) jumps several times between the slow state 0 and the fast state 1.

In JMT, the MMPP(2) is parameterized by four rates: $\sigma_0$, $\sigma_1$, $\lambda_0$, and $\lambda_1$ (see figure C.1). While in state 0, the MMPP(2) generates arrivals with rate $\lambda_0$ or jumps to state 1 with rate $\sigma_0$; in state 1 the arrival and jump rates are $\lambda_1$ and $\sigma_1$, respectively. This can be equivalently stated as follows: in state 0 a random number with exponential distribution having rate $\lambda_0 + \sigma_0$ is drawn, then with probability $\lambda_0 / (\lambda_0 + \sigma_0)$ is the arrival of a new job (or equivalently the completion of its service if the MMPP(2) is used as a service process), while with probability $\sigma_0 / (\lambda_0 + \sigma_0)$ without any job arrival or service completion; the case of state 1 is similar.

![Diagram](figure.png)

**Fig. C.1.** Main structure of the MMPP2 distribution (Source: Figure 3.11 [84])

In our case, we selected an MMPP2 distribution for the user arrival based on Balachandran *et al.* [5]. The distribution parameters are:

- The inter-arrival time ON state: $\mu_{on} = 38s$
- The mean duration OFF state: $\mu_{off} = 6mn$
- The number of users during ON state: $n = 20$
We calculated the four rates for the MMPP2 distribution as follows:

- \( \lambda_0 = 1/\mu_{on} \)
- \( \sigma_0 = \lambda_0/n \) (After \( n \) users, the MMPP2 jumps to the OFF state)
- \( \lambda_1 = 0 \) (There are no arrivals during the OFF state)
- \( \sigma_1 = 1/\mu_{off} \)
List of Figures

2.1 Abstract layered architecture for context-aware systems ........ 11
3.1 Service oriented concepts ............................................. 19
3.2 Standards and specifications for service discovery .................... 20
3.3 Service discovery interactions in Jini ............................... 21
3.4 SLP service discovery scenarios ...................................... 22
3.5 UPnP service discovery interactions ................................... 23
3.6 SDP client-server interaction ......................................... 24
3.7 WSDL 1.1 document structure ........................................ 25
3.8 The Devices Profile for Web Services protocol stack ............... 27
3.9 DPWS message exchanges ............................................. 29
4.1 Design methodology for the distributed architecture ................ 39
4.2 Ontology-based context vocabulary model ............................ 40
4.3 Web service generation process ...................................... 42
4.4 Client activity diagram ................................................ 43
4.5 Provider activity diagram .............................................. 44
4.6 Client/provider interactions sequence diagram ........................ 45
4.7 Prototype architecture ................................................ 46
5.1 Discovery ratio for $P(c, 10, *)$ ....................................... 56
5.2 Discovery ratio for $P(c, *, *)$ ......................................... 57
5.3 Discovery ratio for $P(*, *, *)$ ......................................... 58
5.4 Mean discovered services per request for $P(c, 10, *)$ .............. 60
5.5 Mean discovered services per request for $P(c, *, *)$ .............. 60
5.6 Mean discovered services per request for $P(*, *, *)$ .............. 61
5.7 Probe overhead .......................................................... 63
C.1 Main structure of the MMPP2 distribution .......................... 87
List of Tables

2.1 Summary of discussed context-aware systems ................. 13
3.1 Summary of discussed service discovery approaches .......... 32
5.1 Simulation parameters ........................................ 53
5.2 Summary of discussed user behavior models .................. 54
5.3 Selected statistical distributions for the characterization of
user behavior .................................................. 54
5.4 Discovery ratio measures ...................................... 59
5.5 Mean discovered services per request measures .............. 62
Listings

4.1 Service QName example................................... 41
A.1 Context ontology ........................................ 73
B.1 Context ontology service WSDL document ............. 77
B.2 Example of a solution service WSDL document .......... 78
B.3 Hello message .......................................... 79
B.4 Bye message ........................................... 80
B.5 Probe message ......................................... 81
B.6 Probe match message ................................... 82
B.7 Get device metadata message ............................ 82
B.8 Get device metadata response message .................. 83
B.9 Get service metadata message ............................ 84
B.10 Get service metadata response message ............... 84
B.11 Invoke message ....................................... 85
B.12 Invoke response message ............................... 85
References


68. Hung Q. Ngo, Anjum Shehzad, Saad Liaquat, Maria Riaz, and Sungyoung Lee. Developing context-aware ubiquitous computing systems with a unified mid-
100  References


List of publications


Best poster award at *Les Doctoriales® transfrontalières 2009*

---

Un alumno de la UIB, premio al mejor póster de la tesis del programa de doctorales

Mehdi Khouja es estudiante de doctorado del Departament de Matemàtiques

M.G.

El póster de presentación de la tesis que está realizando Mehdi Khouja, estudiante de doctorado del Departament de Matemàtiques y informàtica de la UIB ha obtenido el premio al mejor póster en el concurso organizado por el programa Doctoriales® 2009, celebrado recientemente en Mallorca. Khouja está integrado en el grupo de investigación de 'Arquitectura i comportaments de sistemes informàtics i de comunicacions' (AC-SIC) de la UIB.

La semana Doctoriales® transfrontalières 2009 es un programa europeo organizado por la Universidad de Perpinyà y la UIB con el objetivo de preparar el póster de la tesis y orientar profesionalmente a los alumnos para su incorporación al mercado de trabajo, su entrada en una empresa y/o para la creación de su propio negocio.

El programa acogió a 80 estudiantes de doctorado de las universidades de Perpinyà, Tolosa, Montpellier, Girona, Barcelona (UB y UPC) y Baleares.

La tesis de Mehdi Khouja consiste en proponer un mecanismo de auto-descubrimiento de los servicios en entornos móviles. Se basa en que en la actualidad, la omnipresencia de dispositivos electrónicos en la vida diaria provoca la necesidad de invertir más tiempo de configuración por parte del usuario. Para que el usuario pueda interactuar y utilizar fácilmente los servicios que se ofrecen en su entorno, hace falta automatizar la tarea de configuración y de descubrimiento de los servicios. FACilitar, por ejemplo, que el usuario en un aeropuerto pueda recibir a través de su móvil información sobre el estado de su vuelo, publicidad de las instalaciones o detectar la presencia de algún conocido a través de los contactos de la agenda sin que tenga que hacer nada.

*Ultima Hora* 29 Oct. 2009
La tesi proposa un mecanisme d'autodescobertiment dels serveis en entorns mobils, basat en el modelatge semàntic del context.

Les doctoriales transfrontalièrtes premien un doctorand de la UIB

Diario de Mallorca 05 Nov. 2009